6-1-1992

The Design and implementation of an 8 bit CMOS microprocessor

Jeffrey Correll

The Design and Implementation of an 8 bit CMOS Microprocessor

by

Jeffrey Correll

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

Approved by:

George A. Brown (Thesis Advisor)

Jong D. Chong

Robert E. Pearsen

Roy S. Gemikowski (Department Head)

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK
JUNE 1992

THESIS RELEASE PERMISSION FORM

ROCHESTER INSTITUTE OF TECHNOLOGY
COLLEGE OF ENGINEERING

Title of Thesis: The Design and Implementation of an 8 bit CMOS Microprocessor

I, Jeffrey A. Correll, hereby refuse permission to the Wallace Memorial Library of RIT to reproduce my thesis in whole or in part.

Date: 6-30-92

TABLE OF CONTENTS

Abstract i

Introduction 1

1.0 Theory 3

1.1 Organization and Behavioral Modeling 12
1.1.1 Control Unit 15
1.1.2 Instruction Register 17
1.1.3 General Purpose Registers 18
1.1.4 Stack Pointer 19
1.1.5 Flags Register 20
1.1.6 ALU 21
1.1.7 Multiplier 22
1.1.8 Memory Data Register 23
1.1.9 Shifter 24
1.1.10 Bus Unit 24

2.0 Logic Circuit Construction 28

2.1 Control Unit 30
2.2 Instruction Register 39
2.3 General Purpose Registers 41
2.4 Stack Pointer 43
2.5 Flags Register 47
2.6 ALU 51
2.7 Multiplier 54
2.8 Memory Data Register 57
2.9 Shifter 60
2.10 Bus Unit 63

3.0 Microprocessor Layout 75

4.0 Conclusions 78

5.0 References 80

TABLE OF CONTENTS [continued]

Appendix A

Control Unit Control Flow Chart A-1

Appendix B

Bus Unit Control Flow Chart B-1

Appendix C

Processor BLM Code C-1

Appendix D

Quicksim Logical Simulation Listings D-1

Instruction Register D-1
ALU D-6

Appendix E

New Library Components E-1

Appendix F

Program Simulation Results F-1

LIST OF FIGURES

1. Block Diagram of Processor 14
2. Control Unit Block Symbol 16
3. Instruction Register Block Symbol 17
4. General Purpose Register Block Symbol 18
5. Stack Pointer Register Block Symbol 19
6. Flags Register Block Symbol 20
7. ALU Block Symbol 21
8. Multiplier Block Symbol 22
9. Memory Data Register Block Symbol 23
10. Shifter Block Symbol 24
11. Bus Unit Block Symbol 27
12. Logic Circuit of Control Unit 31
13. Logic Circuit of Control Unit Positive Clock PLA 32
14. Logic Circuit of Control Unit Negative Clock PLA 33
15. Logic Circuit of Register Control Block 35
16. Logic Circuit of Control Unit Register Decode Circuitry 36
17. Logic Circuit for ALU Control 37
18. Logic Circuit for Jump Control 38
19. Logic Circuit of Instruction Register 40
20. Logic Circuit of General Purpose Registers 41
21. Logic Simulation Results of General Register 42
22. Logic Circuit of Stack Pointer Register 44
23. Results of Stack Pointer Simulation 46
24. Logic Circuit of Flags Register 48
25. Simulation Results of Flags Register 50
26. Logic Circuit of ALU 53
27. Logic Circuit of Highest Multiplier Hierarchy Level 54
28. Logic Circuit of Multiplier Core 55
29. Simulation Results of Multiplier 56
30. Logic Circuit of Memory Data Register 58
31. Simulation Results of Memory Data Register 59
32. Logic Circuit of Shifter 61
33. Simulation Results of Shifter 62
34. Read Cycle Timing Diagram 63
35. Write Cycle Timing Diagram 64
36. Logic Circuit for Bus Unit 65
37. Logic Circuit for Bus Unit PLA 66
38. Logic Circuit for Bus Unit Timer 67
39. Logic for Parallel Load 8-bit Counter 68
40. Logic for 8-bit Tri-State Latch 69
41. Logic for Reset State Latch 70
42. Logic for Prefetch Queue 72
43. Logic for Prefetch Queue Counter 73
44. Logic for 8-bit Tri-State Register 74
45. Microprocessor Layout Floorplan 77
A-1. Control Flow Chart of Control Unit - Pt 1 . . . . . A-1

LIST OF FIGURES [continued]

A-2. Control Flow Chart of Control Unit - Pt 2 ..... A-2
A-3. Control Flow Chart of Control Unit - Pt 3 ..... A-3
A-4. Control Flow Chart of Control Unit - Pt 4 ..... A-4
A-5. Control Flow Chart of Control Unit - Pt 5 ..... A-4
A-6. Control Flow Chart of Control Unit - Pt 6 ..... A-5
A-7. Control Flow Chart of Control Unit - Pt 7 ..... A-6
B-1. Control Flow Chart of Bus Unit - Pt 1 ..... B-1
B-2. Control Flow Chart of Bus Unit - Pt 2 ..... B-2
B-3. Control Flow Chart of Bus Unit - Pt 3 ..... B-3
B-4. Control Flow Chart of Bus Unit - Pt 4 ..... B-4
E-1. Logic Symbol of Trilat Circuit E-1
E-2. Transistor Level Schematic of Tri-State Latch for Logic Simulation E-2
E-3. Logic Simulation Results of Trilat E-2
E-4. Transistor Level Schematic of Tri-State Latch for Physical Simulation ... E-3
E-5. Simulation Results of Trilat E-4

LIST OF TABLES

1. ALU Register-Register Instruction Format 4
2. ALU Immediate Addressing Instruction Format 5
3. ALU Direct Addressing Instruction Format 5
4. ALU operations 6
5. Shifter operations 7
6. Miscellaneous operations 7
7. General Instruction Format 8
8. Shifter Instruction Format 10
9. ALU Truth Table 51

Abstract

The design and implementation cycle of an 8 bit CMOS microprocessor is discussed. The primary steps in the design procedure of the microprocessor consists of instruction selection, instruction encoding and organizational specification. A simple architecture is chosen to allow the emphasis of this investigation is focused upon the entire design procedure,

Software behavioral models of functional blocks within the processor are used to validate the architecture. The functional blocks are then replaced with logic circuit models and tested.

After logical simulations of all blocks have been completed, physical simulations of the logic circuits are performed using a SPICE like simulator to extract delay characteristics of longest circuit paths. Using this delay information, a preliminary estimate of processor speed is possible.

Layout of the processor is generated using the Department of Computer Engineering's 2 μM CMOS Standard Cell Library.

Introduction

Because of the sheer number of devices and the complexity of today's monolithic microprocessors, techniques have evolved to effectively manage their design and implementation.

In the early days of microprocessor development, it was common to build a hardware prototype of the processor before integrating it onto silicon. The prototype was built to enable the designers to validate proper system operation. With a processor of 5,000 gates for example, this was no small undertaking. One small change in logic could very well effect the operation of the rest of the system. Basic operational problems also exist with prototype implementation. How could one be sure that the problem with the circuit was a design flaw and not a broken trace, a loose wire wrap or a faulty part? Also will the integrated circuit design operate properly or with the same characteristics as the discrete hardware prototype?

After the design was validated, a physical layout drawing was created in order fabricate the circuit on silicon. This was accomplished by using rulers and pencils to draw the various masks on graph paper at a draftsman's table.

By today's standards however, a prototype is not only unwise but virtually impossible to realize. With the number of devices on a single chip exceeding the 2 million mark, a discrete model would occupy a significant amount of area as well as dissipate enough energy to heat a moderate size room. Also, the basic implementation problems would still exist except that they would be exacerbated by several orders of magnitude. The solution lies in using computers to model computers. More specifically, in creating software that will behave like the proposed architecture.

Once the architecture of a system has been established, it is possible to write software to model its behavior. Intuitively enough, this type of software is referred to as a behavioral model. Much of the time the architecture is broken down to reveal some of the proposed organization as well. This is done in order to partition the processor into functionally independent sections. The interface between individual parts of the microprocessor can then be established and tested. This encourages a modular design wherein techniques and even circuitry within a particular module may be changed but not affect the operation of other sections, as long as the interface remains constant.

After validation of the architecture has been completed, it is possible to start creating logic circuit models to replace the behavioral models. This technique has several advantages. Not the least of which is that by replacing behavioral modules individually, the logic circuit designs may be tested with a 'perfect' environment. What this means is that if the processor simulation operated according to specifications with the behavioral software models, then the replaced module can be the focus of attention, and (hopefully) any problems encountered will be the result of the logic model. This procedure is repeated until all modules in the design are composed of logic models.

After simulations of the logic models insure proper system operation, the simulations of the physical circuit models must be made to verify that the device sizing is adequate to drive the attached loads in the specified periods of time.

1.0 Theory

The design and implementation procedure outlined in the Introduction section is essentially the one followed in the completion of this microprocessor.

The first step was to create the architecture for the microprocessor. Because this was an experiment in the design process as a whole, and not one devoted strictly to computer architecture; a simple 8 bit architecture was chosen. The first step was to write down all the instructions that were desirable to have in a microprocessor from a software engineer's point of view. This list was successively reduced over several iterations leaving the list designated Table 4 as the ALU related operations, Table 5 as the shifter operations and Table 6 as the miscellaneous instructions.

The criteria for exclusion of instructions was based upon their functionality in executing a reasonable program and intuition as to their difficulty of implementation. The designer's conception of the format of the system organization and that fact that this entire project was being done by him in the time span of approximately 1 school year also played an important part in limiting the size of the architecture.

As one can plainly see in Tables 4-6 the list of instructions is simple and contains nearly all the basic operations one would expect to find in any general purpose processor on the market today. The columns define the instruction mnemonic, the actual op-code for various addressing modes, the PSW (**P**rocessor **S**tatus **W**ord) , and a simple description of the operation.

The PSW is a nibble containing flags NZVC (**N**egative, **Z**ero, **O**verflow and **C**arry). These flags report the status of various operations performed by the processor and allow for all standard conditional jumps.

Depending upon the instruction, up to three different addressing modes could apply. The first mode is register-register addressing. This is used when an instruction deals only with data that is contained within registers. This mode applies to all of the ALU related operations, the shifter operations and the multiply instruction.

Because the ALU operations may be used with any of the three addressing modes. the General Instruction Format shown in Table 7 has three specific forms that correspond to these addressing modes. For the register-register addressing mode, the instruction format is as follows:

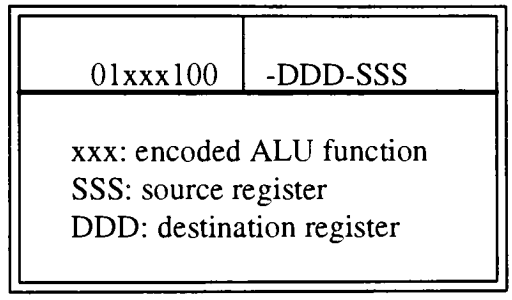

TABLE 1 ALU Register-Register Instruction Format

The next mode is immediate addressing which causes a constant 8-bit value to be used in an operation. This mode is specified with the ALU instructions, load or store. The format for ALU instructions in this mode is given by:

01xxx000	-DDD----	ZZZZZZZZ

xxx: encoded ALU function
DDD: destination register
ZZZZZZZZ: immediate data

TABLE 2 ALU Immediate Addressing Instruction Format

The last mode available is the direct addressing mode. Direct addressing requires that along with the operation code the instruction must contain an 8-bit address where the desired data resides.

01xxx010	-DDD----	ZZZZZZZZ

xxx: encoded ALU function
DDD: destination register
ZZZZZZZZ: address of data

TABLE 3 ALU Direct Addressing Instruction Format

The load and store instructions have only the immediate and direct addressing modes associated with them. A register-register load or store has no meaning in this processor.

Mnemonic	Opcode	Flags NZVC	Description
ADD	$40 immed. $42 direct $44 reg-reg	XXXX	2's complement Addition
SUB	$48 immed. $4A direct $4C reg-reg	XXXX	2's complement Subtraction
NOT	$50 immed. $52 direct $54 reg-reg	XX00	Logical Inversion
AND	$58 immed. $5A direct $5C reg-reg	XX00	Logical AND
OR	$60 immed. $62 direct $64 reg-reg	XX00	Logical OR
XOR	$68 immed. $6A direct $6C reg-reg	XX00	Logical Exclusive-OR
NEG	$70 immed. $72 direct $74 reg-reg	XXXX	2's complement Inversion
CMP	$78 immed. $7A direct $7C reg-reg	XXXX	Byte Comparison

TABLE 4 ALU operations

Mnemonic	Opcode	Flags NZVC	Description
LSR	$C0	0X0X	Logical Shift Right
LSL	$C8	XX0X	Logical Shift Left
ASR	$D0	XX0X	Arithmetic Shift Right
ASL	$D8	XX0X	Arithmetic Shift Left
ROR	$E0	XX0X	Rotate Right
ROL	$E8	XX0X	Rotate Left
Push	$88	----	Push Register
Pop	$80	----	Pop Register

TABLE 5 Shifter operations

Mnemonic	Opcode	Flags NZVC	Description
LD	$08 immed. $0A direct	----	Load Register
ST	$10 direct $12 indirect	----	Store Register
MULT	$98 reg-reg	----	Multiply (c:d <- a*b)
SCB	$B0	---1	Set Carry Bit
CCB	$A8	---0	Clear Carry Bit
JMP	$B8 $B9 $BA $BB $BC	----	Jump Always Jump if equal/zero Jump if overflow/underflow Jump if carry/borrow Jump if negative
Halt	$A0	----	Halt
NOP	$00	----	No Operation (Idle Cycle)

TABLE 6 Miscellaneous operations

Table 7 shows the general instruction format devised for the processor. Each of the fields of the instruction are listed and named:

TTxxxRM-	-DDD-SSS	ZZZZZZZZ

TT: operation type
xxx: instruction specific field
R: register-register bit
M: data modifier bit

DDD: destination register
SSS: source register

ZZZZZZZZ: data/address byte

TABLE 7 General Instruction Format

Some explanation of the possible values of each field and their uses is in order:

TT: the two bit operation type field is used to distinguish which of the general types of operations the op-code belongs to. The basic types are 00 which corresponds to a NOP, LOAD or STORE, 01 and 11 which indicates that the encoded data bits are sent directly as control signals to the ALU and shifter respectively, and lastly 10 is used to represent miscellaneous instructions whose bits do not correspond to any special control line encoding.

xxx: these three bits determine which specific instruction is to be executed within a specific operation type. For instance if the TT bits are 01 and these bits are 000 this would indicate an ALU add operation since setting the control lines of the ALU to 000 would cause an add to be performed.

R: this bit designates if the op-code is a register-register operation (**R=1**) or a register -memory function.

M: the data modifier bit works in conjunction with the **R** bit to more specifically determine which type of addressing is used. When register-register addressing is used this bit has no significance but should be set to 0. However if register-memory addressing is used, an **M=0** would indicate immediate addressing and **M=1** represents direct addressing.

DDD: this field of the second byte determines where the results of an operation is to be placed. It also specifies the first data source in register-register and register-memory operations.

SSS: this field is used to specify the second data source in register-register operations. It only becomes active when **R=1**.

ZZZZZZZZ: used to hold the data for an immediate data operation or the address for a direct addressing mode function.

One additional note on the subject of instruction encoding. Since all of the shifter related functions are by definition single register functions, the encoding scheme for these is slightly different. Table 8 shows the modified format for shifter operations. The only differences are that the instruction occupies only one byte and the source/destination register is encoded in the three least significant bits of the instruction.

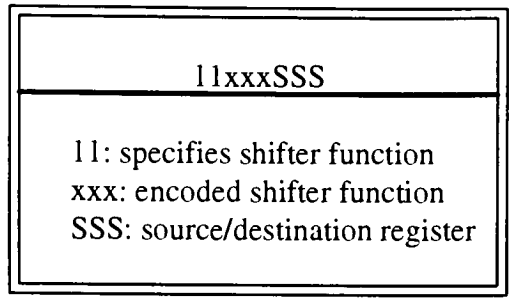

TABLE 8 Shifter Instruction Format

Although there are truly only three addressing modes in which to deal with data in operations, there are some instructions that do not use any of the aforementioned addressing modes. These instructions could be classified as using inherent addressing (**TT=01**) due to the nature of their operations.

One such instruction is multiply. This instruction has the source and destination registers hardcoded into it. The way in which multiply works can be explained by the following RTL statement:

$$C{:}D \Leftarrow A*B$$

This equations states that the A register is the multiplier, the B register is the multiplicand, and the C and D registers hold the most significant and least significant bytes of the results respectively.

The instructions for Set Carry Bit (SCB), Clear Carry Bit (CCB), Jump, and Halt also reside in the inherent category. All of these instructions have only one form, except for the jump instruction which requires some explanation. As one can observe from Table 6, the jump instruction has several forms which correspond to a jump on certain conditions. In order for the instruction to jump to a new address, the address

provided in the second byte of the jump instruction in the form of an eight bit number which corresponds to the absolute address of the destination.

The processor's address space was restricted to eight bits. This was done in order to keep the address busses and data busses the same size, thus reducing the complexity of the processor. If the expansion of the address space to 16 bits or more is desired, the same design techniques could be followed, and the same architecture used, with some changes to accommodate the larger sizes.

If the word size of the processor and the address bus size are the same, then the only changes that need to be made are that all registers that hold data or addresses and all of the data and address busses must be made large enough to accommodate the new word size.

If the address space is to be increased by a different amount than the word size, then major design changes will have to be made. This will involve determining how an address will be fetched from memory on the data bus. One technique would be to read in portions of the address over several cycles. For instance if the word size is to remain eight bits but the address space is increased to a 32 bit size, then an address could be acquired from memory in four reads (one read per byte).

There also remains the problem of reading this address into an internal register such as the Stack Pointer. One solution would be to have an internal data bus of 32 bits. During a data operation only one byte of the bus is used.

Since the address and data busses of this microprocessor are the same, the problems just mentioned will not be addressed in detail.

1.1 Organization and Behavioral Modeling

The organization of the processor was developed in the later half of the architecture specification. It includes partitioning the processor into several functional blocks. These blocks along with descriptions of their functions and diagrams are presented here.

The first step in the implementation of this processor was to model each of the processor's modules in software. Two languages were examined for this purpose. The first was VHDL (VHSIC Hardware Description Language). After consideration, this option was rejected because the release of the VHDL system available from RIT's Department of Computer Engineering required a slightly different version of the simulator than the one used by the basic EDA tools. This would prevent the simulation of VHDL models with circuits using the RIT Department of Computer Engineering's Standard Cell Library. Also, because this VHDL was an Alpha release from Mentor Graphics it was not a full implementation of the language. It was felt that a significant amount of effort would be required to overcome problems associated with this version of the software that could be used instead to concentrate on the project at hand.

The language that was chosen was Mentor Graphics' BLM (Behavior Language Model) or more specifically C with functions calls and other 'hooks' into Mentor Graphics' Quicksim Logic Simulator. This proved to be a good choice because of the author's familiarity with C, and the ease with which one could construct these models.

After the organization had been decided upon, the interfaces to the various modules were available. The first step in constructing a BLM was to create a symbol for the part in SYMED (Mentor Graphics Symbol Editor) symbol that was graphically meaningful. For instance the BLM for the ALU was constructed to take on the standard 'V' like shape (Figure 7) normally associated with an ALU. These symbols could then be interconnect with NETED, the Mentor Graphics' Schematic Editor, to interface the

modules with one another.

In the descriptions of the processor that are presented here, all control signals are to be considered active high signals unless otherwise specified.

An important point that must be stated here is the author's use of reading and writing as they apply to functional blocks and circuits within the processor. Because the functional blocks were created and modeled on a individual basis, the use of the terms read and write is different than the one normally used. For instance a register is said to write when it presents its stored data on an output bus. Similarly, a registers reads when it captures data at one of its inputs. This convention is maintained throughout the entire processor description.

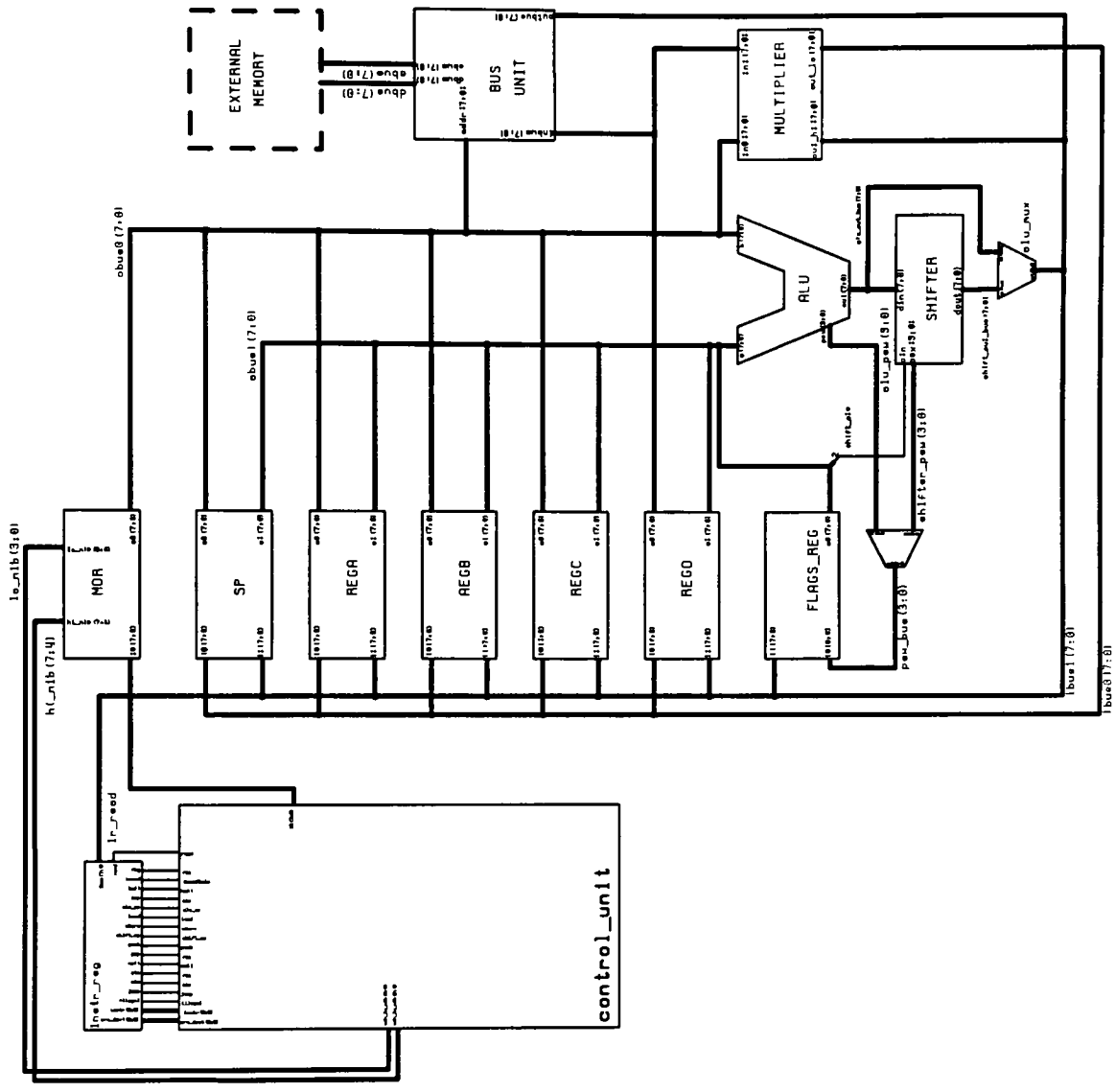

Figure 1 Block Diagram of Processor

1.1.1 Control Unit (CU)

This block performs as a Mealy state machine, therefore the BLM that was written to emulate it was created with a memory of its current state, a knowledge of its input values and the execution steps it should take according to the Control Unit Control Flow Chart found in Appendix A.

If one reads the next few sections describing the input and output pins of the remaining blocks, a good understanding of most of the pins on the CU can be gained. However, there are still a few pins that require explanations:

illegal_instr: This output is raised when an illegal instruction occurs.

read: This output is used to indicate when the instruction register must read its input bus..

i0(7:0): By directly connecting the CU to the i0 bus, the PSW from the flags register is used during a conditional jump, to determine if the condition is true or false.

psw_mux: This is a control signal connected to a multiplexer which chooses the PSW output from the ALU or the shifter and routes it to the Flags Register Input.

mux_sel: This is also a control signal to a multiplexer except that this chooses between the output of the shifter or the ALU.

mux_oe: This control signal enables or disables the multiplexer's tri-state output from the ALU or shifter to the i0 bus.

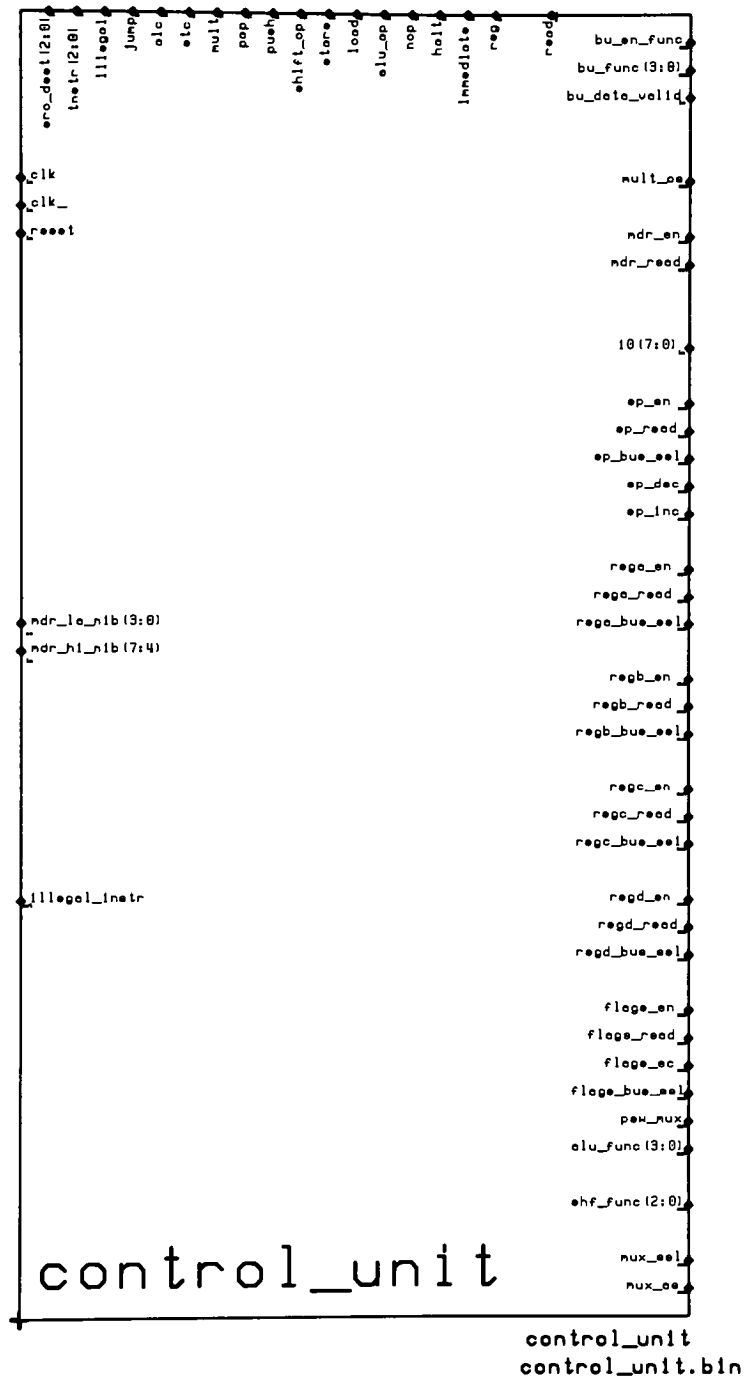

Figure 2 Control Unit Block Symbol

1.1.2 Instruction Register (IR)

This register is a read-only register that accepts an instruction byte and decodes the operation of that byte. It provides the control unit with information such as instruction type (ALU operation, load, store, shift, etc) as well as addressing mode (immediate, register-memory, register-register)

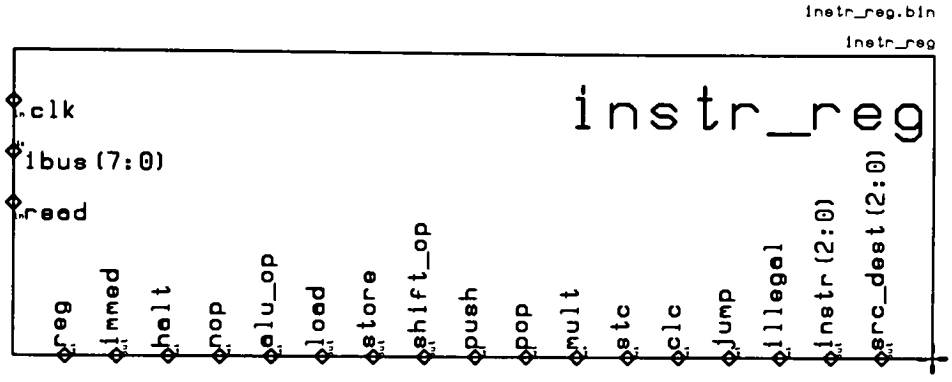

Figure 3 Instruction Register Block Symbol

The decoding of opcodes read from memory is performed by combinational circuitry attached to the read-only register. The BLM decoding was performed in a straightforward manner by reading in data from the *ibus* input when the *clk* and *read* inputs were both high. After the data was acquired, the byte was cast into a C union structure so that individual bit fields could be addressed. This would allow for simple conditional statements to be used in determining how the output lines should be set for the CU.

The determination of what output lines the instruction register would have was decided during the instruction encoding phase of design. The output lines can be directly associated with different cases discussed in the instruction encoding section.

For instance when the *reg* output is high it indicates a register-register instruction. Similarly if the *alu_op* line is high then the bits contained in the *instr(2:0)* bus are routed directly to the ALU as control signals. All of the other signals are fairly obvious with the exception of *src_dest(2:0)* which is the source/destination register specification for a shift register instruction.

1.1.3 General Purpose Registers

These 4 registers (denoted A,B,C and D) are used as intermediate locations to hold results of operations. They are read/write registers having access to all four internal data busses.

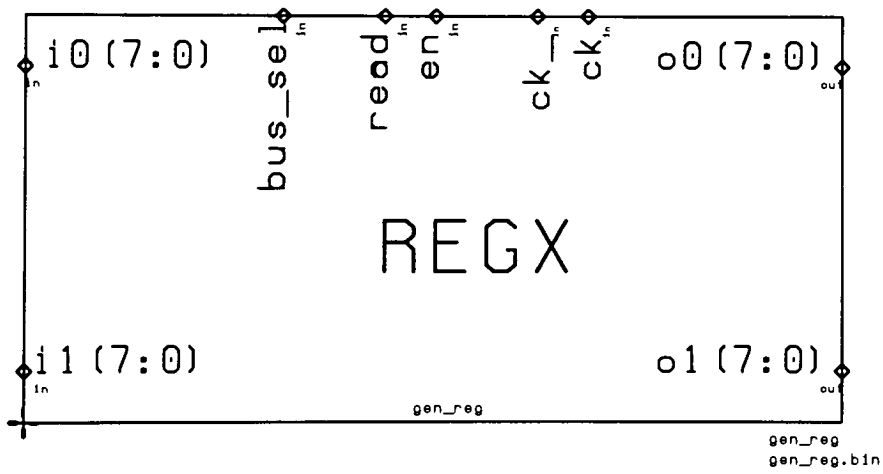

Figure 4 General Purpose Register Block Symbol

Since all of registers in the processor were to operate similarly, the BLM code (and later the logic circuit schematics) was simply copied and altered to suit the individual requirements of specific registers. The construction of the General Register models

was done in this way; using the basic code from the Instruction Register, modifications were made to include two input and two output busses, a pin for selecting the bus (a or b), a pin for selecting the mode (a high on the read line would read an input bus, a low would write to an output) and an enable pin.

1.1.4 Stack Pointer (SP)

The Stack Pointer Register is used to maintain the address used in pushes and pops (and also in subroutine jumps if implemented). This register is essentially the same as the General Purpose Registers with the exception that it can automatically increment or decrement the value it has stored with the application of a high signal on the *inc* or *dec* inputs respectively.

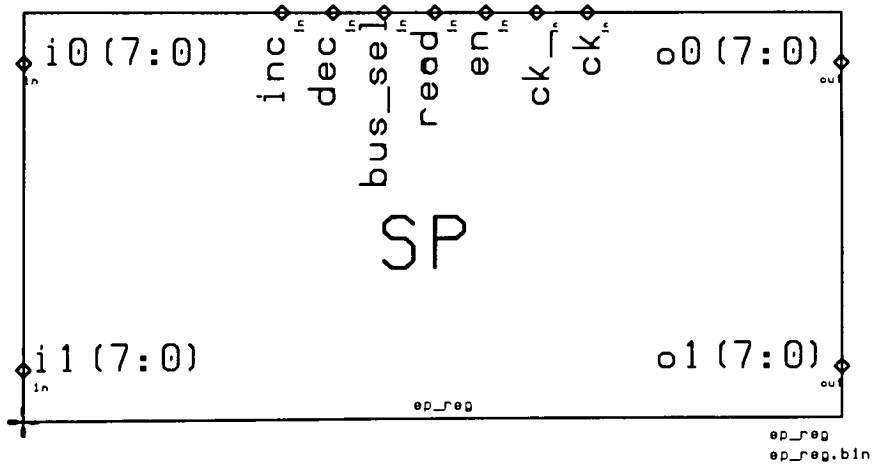

Figure 5 Stack Pointer Register Block Symbol

1.1.5 Flags Register (Flags)

The Flags Register was designed to maintain the four PSW bits and have the ability to set or reset the carry bit on command. Although the input *i1* and output *o0* are shown to be eight bits wide this was done only to make the register compatible with the existing eight bit wide internal data busses.

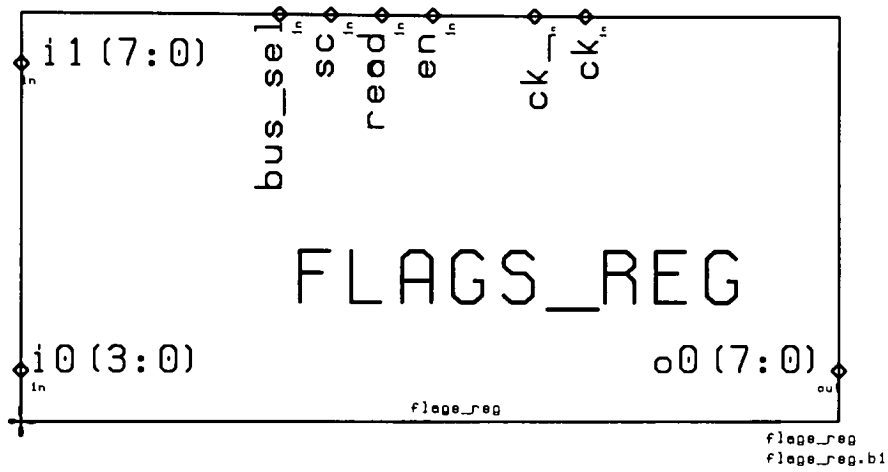

Figure 6 Flags Register Block Symbol

It is plainly obvious that the Flags Register has only one output, this was done because the only place that the PSW need go was to memory to be written as data and through the ALU during a CMP instruction. The lack of another output bus was also advantageous since it allowed for an unused combination of the register's inputs to serve as the means of setting or resetting the carry bit. When the *bus_sel* input was set high and the *read* input was low (thus attempting to write to a nonexistent output bus *o1* the register would assign the value of the *sc* (set carry) input to the carry bit in the register.

1.1.6 Arithmetic Logic Unit (ALU)

The ALU was conceived to be a generic, combinational circuit having 2 data bus inputs (*a* and *b*) and one resultant data output. Also present was a 4-bit control bus to select among the nine available functions, and a 4-bit bus output containing the PSW.

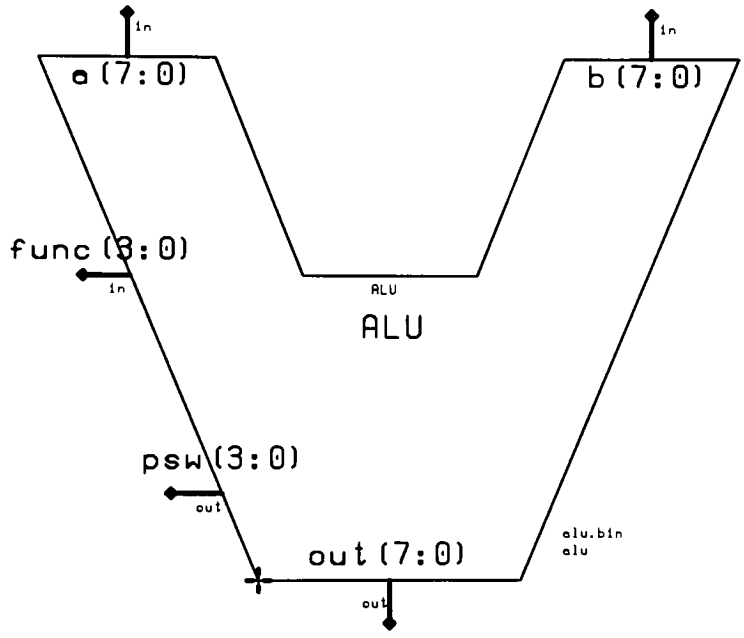

Figure 7 ALU Block Symbol

The nine functions that the ALU will perform can be found in Table 4 with the addition of a pass_a and pass_b modes that will pass the a and b inputs respectively through to the output. If one looks at the ALU instruction chart, it is apparent that there is no ADD instruction with a carry in. This was excluded in order to make the ALU more simple.

The PSW output of the ALU will change whenever data is presented at the inputs of the ALU. However, this output will not be stored in the Flags register until the

appropriate control signals are applied. In this way, the PSW output may still be referred to as the Processor Status Word since it is not stored on a cycle to cycle basis but rather on an instruction basis.

Upon completion of the ALU specifications, the C code for the block was written and may be found in Appendix C.

1.1.7 Multiplier (Mult)

The multiplier was the simplest of the BLMs to create. It simply multiplied the values of both inputs together and wrote the high byte to the *out_hi(7:0)* output and the low byte to the *out_lo(7:0)* output when the *oe* input was high. Since the multiplier was a combinational circuit, the BLM was written to emulate a combinational circuit. This was done by having the multiplier recalculate the output whenever any of the inputs changed.

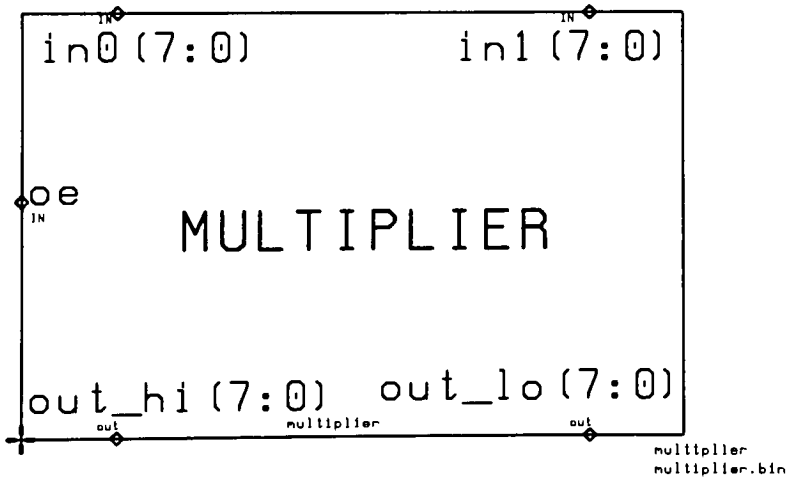

Figure 8 Multiplier Block Symbol

1.1.8 Memory Data Register (MDR)

This register has read/write access to all 4 data paths as well as write capability of high and low nibbles to the CU. The purpose of the high/low nibble write is to allow the CU to decode the register(s) involved in operations.

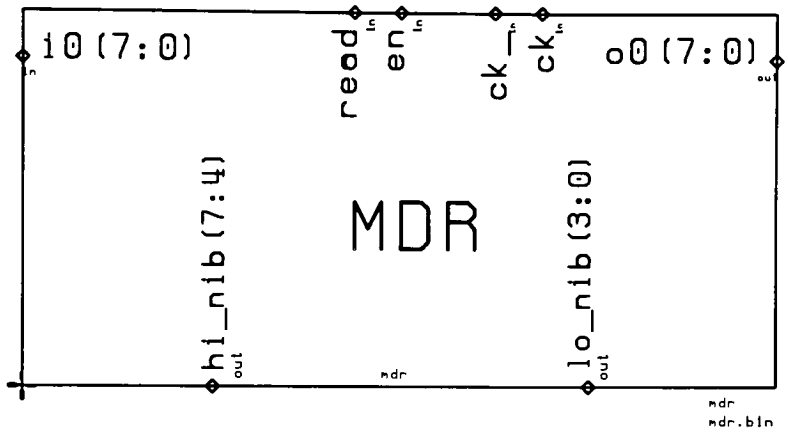

Figure 9 Memory Data Register Block Symbol

The MDR again used the basic register BLM code. The exceptions this time were that the MDR had only one input and one output bus thus eliminating the need for a *bus_sel* line. However there were two additional four bit output busses added: *hi_nib(7:4)* and *lo_nib(3:0)*. These two busses were used to transport the hi and low nibbles of the second byte in an op-code (that which contained the source and destination registers when applicable) to the Control Unit.

These two additional busses were easily implemented. By simply using logical bit masks and shifts the appropriate nibbles of the store data byte could be separated and sent out on the correct bus.

1.1.9 Shifter

The next block of the processor to be created was the shifter. Since all of the functions in the data path of the processor were to be completed in 1 clock cycle, it was decided that a shifter using switching techniques would be used. It would have a data input and output as well as inputs for control lines and an PSW output. Although Figure 10 shows a four bit wide PSW bus, the only pertinent signals are **N**, **Z**, and **C** since an overflow will never occur in the shifter.

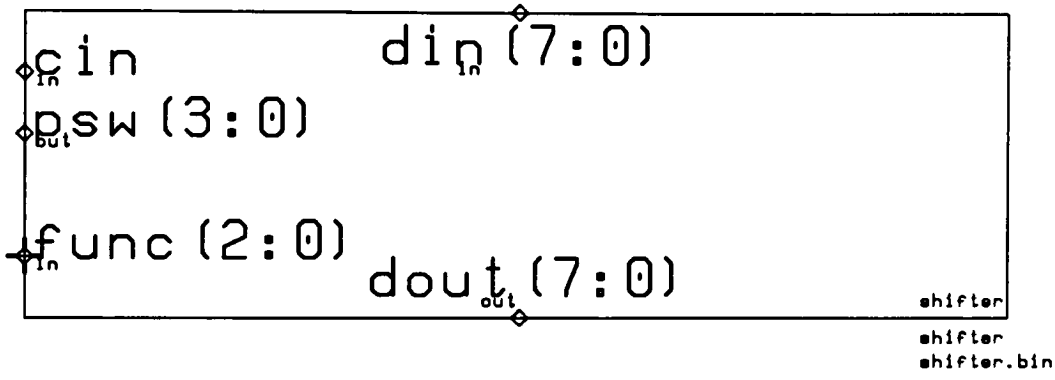

Figure 10 Shifter Block Symbol

The shifter block diagram also shows a carry-in input (*cin*) which is used to input the value of a carry bit for a ROL or ROR operation. This carry bit is provided by the Flags Register.

1.1.10 Bus Unit (BU)

This module was the most complicated model to create and to debug due mainly to its functional complexity. It is a state machine in itself responsible for interfacing with

the external environment.

This module also maintains the PC due to the fact that all addressing operations are absolute and not offset from the PC or any other register. Within the BU is a hardware queue four bytes deep that is used to store instructions that have been fetched from memory at the location specified by the PC. When not synchronized with the CU to perform some specific task, the BU attempts to keep its prefetch full.

Upon system reset, the Bus Unit will bootstrap the processor from a hardcoded memory address. When a reset occurs, the bus unit will automatically read address $00 and fetch that byte which is actually a pointer to the first instruction of the program code to be executed.

The inputs and outputs are described as follows:

inbus(7:0): This input bus is used to present data that is to be written into memory to the bus unit.

addr(7:0): This input bus is used to provide the proper address information to allow the bus unit to write data to, or read data from external memory. This bus is also used to load the Bus Unit's PC register with an instruction address.

outbus(7:0): This bus takes the data from the bus unit to other parts of the processor. The data carried on this bus is from a memory read or a request to the bus unit for the next op-code in the prefetch.

func(3:0): The bus unit has five possible requests that can be made of it. These include:

jump_taken: This function informs the bus that a jump was taken. It should clear the bus units' prefetch queue and accept the address on the *addr* bus as the new memory location to begin an instruction fetch.

write: The data on the *inbus* is written in the location specified by the *addr* bus.

read: The data at the address specified by the *addr* bus is read and placed the it on the *outbus*.

next_opcode: The control unit is requesting the next op-code in memory. If the bus_unit has the data in its prefetch, it immediately presents it on the *outbus* and raises the *data_valid* signal. If the data is not in the prefetch, the BU makes the CU wait unit it can fetch it from memory and then raises the *data_valid* signal.

direct_fetch: The BU takes the first byte in the prefetch and uses it as the address from which to read a data byte. If the prefetch is empty, then the BU gets the next byte and then does the fetch.

en_func: This signal enables a command given to the bus on it on the *func* bus.

data_valid: This output signal is used to inform the CU that the bus unit is presenting the data requested on the *outbus*.

dbus(7:0): The bi-directional data bus to the memory chip.

abus(7:0): This is the address bus to the memory chip.

$\overline{oe}$: The control signal that allows the memory chip to write to the *dbus*.

$\overline{we}$: This signal causes memory to latch the *abus* and *dbus* during a memory write.

reset: This signal informs the BU that a reset condition is in effect and that it should proceed to do a direct_fetch on memory address $00 to get the first instruction.

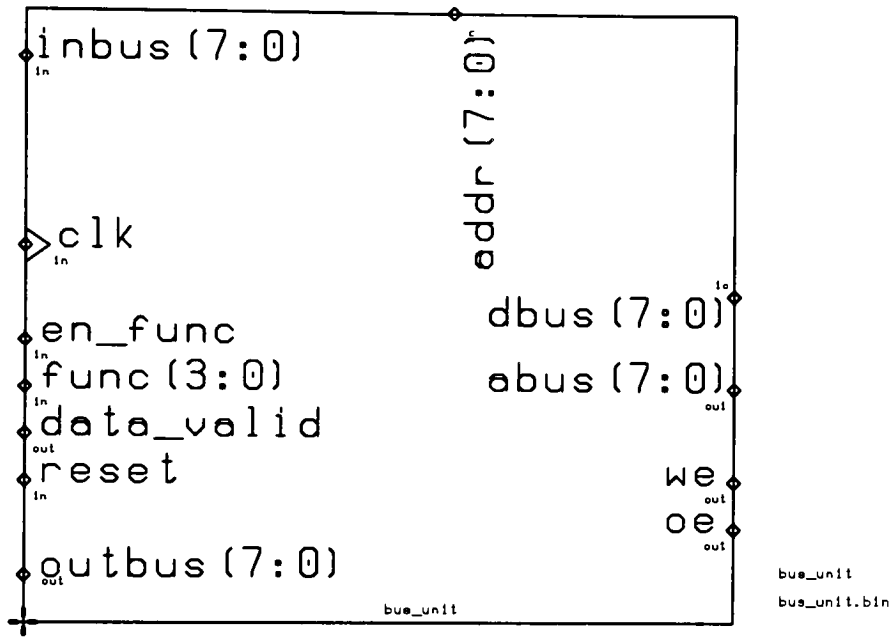

Figure 11 Bus Unit Block Symbol

The bus unit also offers several features that make it unique. One feature involves the Direct Fetching operations. When an instruction is in the direct addressing mode, the next byte in memory is the address of the data. The bus unit has the capability of automatically taking the first instruction from the internal prefetch and writing it to memory as the address. This prevents the needless reading and writing of a memory address register. An Control Flow Chart of the Bus Unit is located in Appendix B.

In addition it was decided that the processor would have four 8-bit data buses two of which would connect the register outputs to the ALU, multiplier and the BU and

two that would connect the outputs of the ALU, multiplier and BU to the register inputs.

2.0 Logic Circuit Construction

Once the BLM simulations were completed and the architecture and the organization were validated by running a variety of test programs, the functional blocks of the processor were replaced with logic circuit models.

The general testing philosophy was that blocks would be constructed from logic parts to functionally emulate the BLMs. The new logic circuits would be thoroughly tested in a stand alone fashion. Once this segragated testing was completed, the new hardware block would be placed into the software model of the entire processor and the test vectors re-run.

It was originally intended to use this procedure until all the blocks of the processor were replaced and a model composed of only hardware could be tested. However, due to the large number of logic circuits in the design and the relative inefficiency of circuit models compared to software models, it was decided that only a few logic blocks would appear in the design at any one time. Although this approach may not uncover all of the problems associated with an all-logic version of the processor, it was felt that the results would be acceptable due to the constraint that the processor's fabrication was not imminent.

After logic simulation of each the circuit blocks was completed using Mentor Graphics' logic simulator Quicksim, the circuits were physically simulated using a SPICE like simulator also from Mentor Graphics called Accusim. This physical simulation was done in order to extract the delay times of the components thus

providing an estimation of maximum clock speed for the processor.

A careful examination of the logic circuit design was used to determine the longest delay paths. The delay time is designated as the elapsed time from 50% of final value of the input signal to 50% of final value of the latest output signal. To simulate loading, an external capacitor of 0.5 pF was placed on the output lines. The simulation temperature was 27° C.

The logic parts (and layout cells) used in this processor were taken from the RIT Department of Computer Engineering's 2 μm CMOS Standard Cell Library created by Larry Rubin. They will henceforth be referred to as The Library. Those parts that were required but not present in The Library were completely developed and tested. These cells will be discussed in detail in Appendix E.

The following is a brief description of The Library components used that are not immediately obvious:

buf - buffer circuit implement with two inverters
cmux2 - 2 input mux
cmux4 - 4 input mux
crxfr - CMOS resistive transmission gate
cxfr - CMOS transmission gate
dff - D-type Flip-Flop
dffar - D-type Flip-Flop with asynchronous reset
dffarcr - D-type Flip-Flop with asynchronous reset; active high clock
dffarscr - D-type Flip-Flop with asynchronous reset and set; active high clock
dffascr - D-type Flip-Flop with asynchronous set; active high clock
dlat - D latch
dlatar - D latch with asynchronous reset
ietri - tri-state inverter

2.1 Control Unit

The largest of the functional blocks to be implemented was the Control Unit. It's large size is due chiefly to the sizable PLA implemented to produce the many signals needed to control the other blocks in the processor.

The input signals to the Control Unit are relatively few (compared to the number of outputs). The majority of these signals originate in the Instruction Register.

The schematic in Figure 12 is the top level hierarchical diagram of the Control Unit. Upon examination of this schematic, one immediately notices that there are several large functional blocks present.

The block designated PPLA (**P**ositve clock edge **PLA**) is the PLA and associated latches used to generate most of the signals that occur in the control unit. A more detailed description of the contents of the PPLA block is found in Figure 13. This figure actually only consists of the transistor logic for the PLA (POS_PLA) and the latches used to capture the output signals when they are valid. The internal circuitry of the POS_PLA block will not be shown because the sheer size of the circuit precludes any detail from being shown when printed.

The NPLA (**N**egative clock edge **PLA**) is used to generate control signals on the negative clock edge. This block is architecturally the same as the PPLA block with Figure 14 showing the block diagram containing the transistor level logic of the PLA (NEG_PLA) and the signal latches. The transistor schematic of the NEG_PLA is also too dense to show much detail, so again, this schematic is excluded.

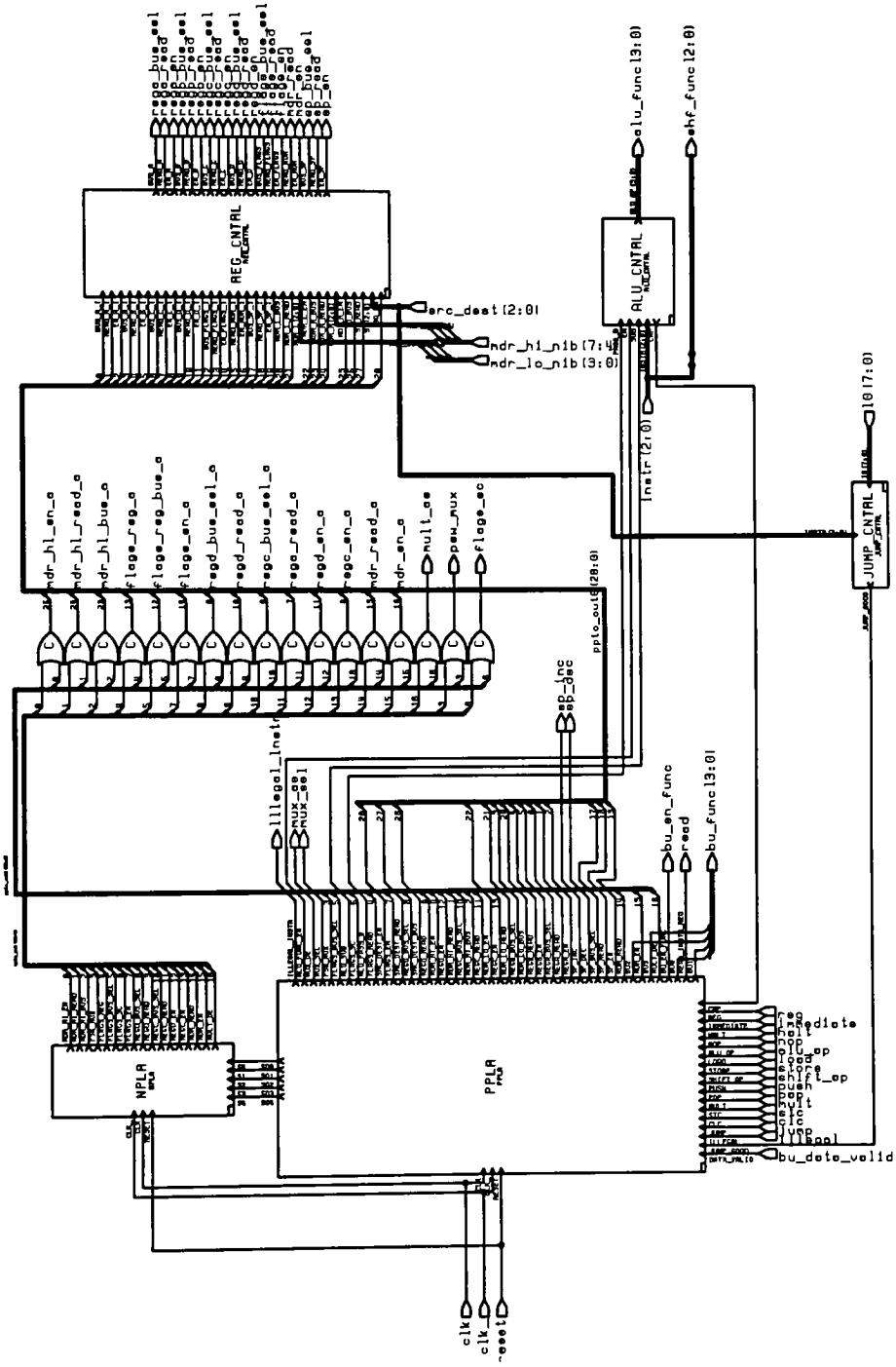

Figure 12 Logic Circuit of Control Unit

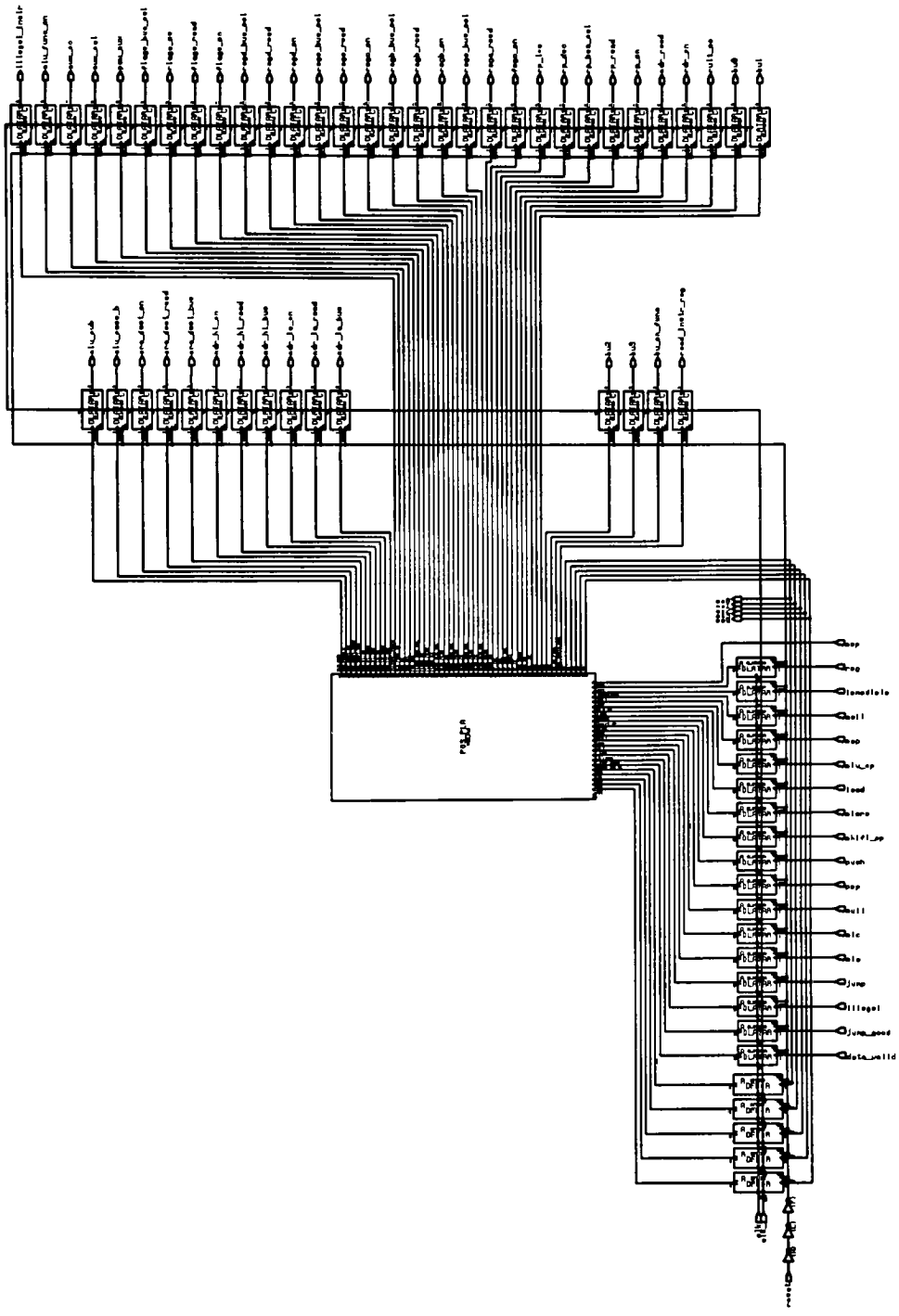

Figure 13 Logic Circuit of Control Unit Positive Clock PLA

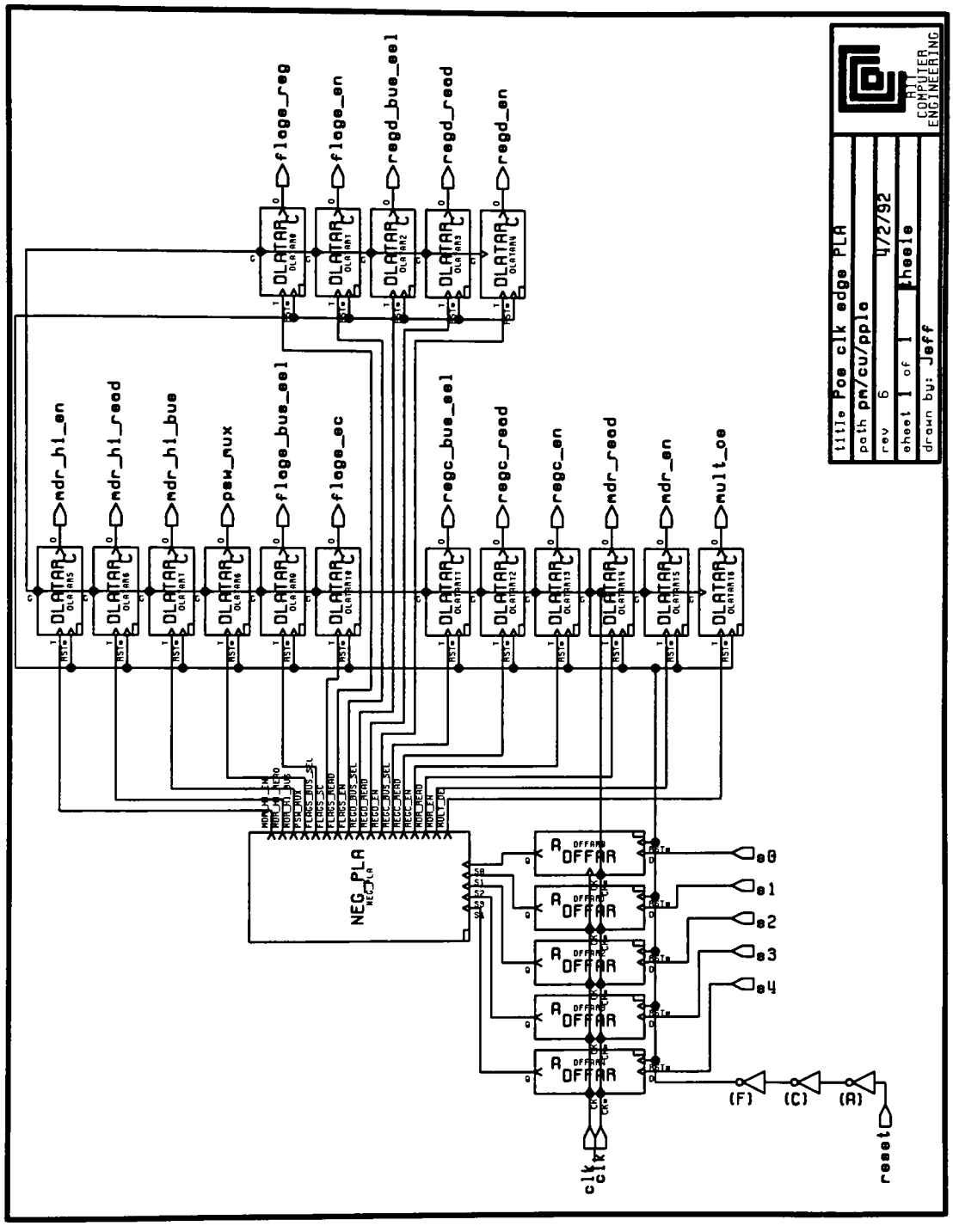

Figure 14 Logic Circuit of Control Unit Negative Clock PLA

The next most sophisticated block in the design is the Register Control (REG_CNTRL) logic. This block is used to control the operation of the General Purpose Registers, MDR, SP, and the Flags Register. Since it is required that these registers be operated from the Control Unit itself, or from another source such as the source/destination byte of a register-register instruction a scheme of multiple control points for each register had to devised. The REG_CNTRL block serves this purpose. At the top of the Figure 15, one notices that there are *en*, *read*, and *bus_sel* (where applicable) lines for each of the registers that are fed directly into OR gates whose outputs go directly to the respective registers. These inputs are fed from the POS_PLA and NEG_PLA directly to operate specific registers at a particular time (such as the hardcoded register designation during a MULT operation)

In the middle of the schematic are located three functional blocks called REG_DECODE (Register Decode). These blocks take the register encoded bits that appears in the source/destination bye or the last three bits in the shifter instruction and decodes them to operate the processors registers. Each of these blocks also have an enable line which controls whether or not the encoded bits the blocks represent are decoded and passed out of the block. A logic level diagram of the REG_DECODE block may be found in Figure 16.

This technique of register decoding relies on the assembler used with the processor to ensure that first, no reference is made to a nonexistent register and second, that a conflicting signal to a register does not occur (such as one register being told to read and write at the same time).

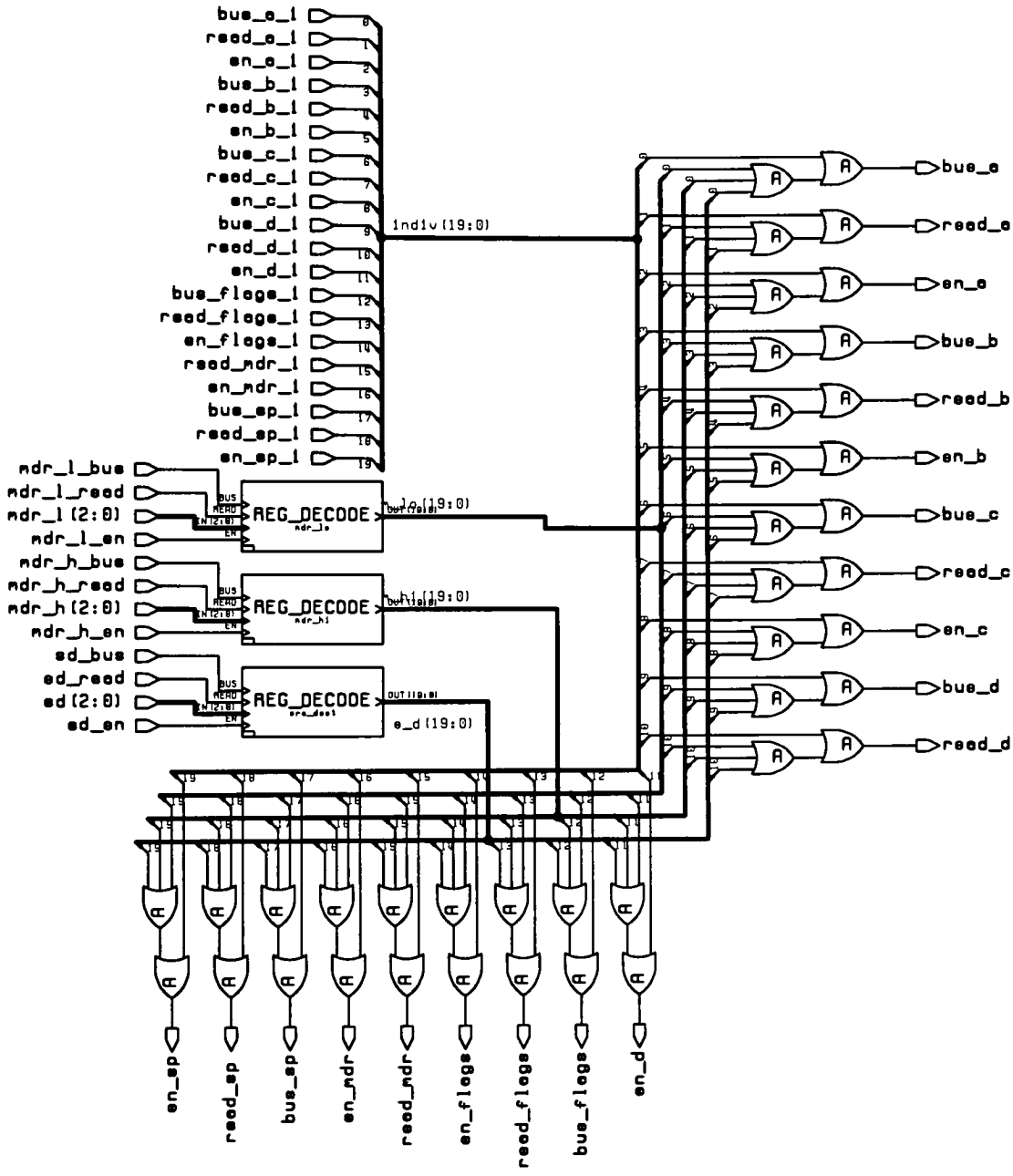

Figure 15 Logic Circuit of Register Control Block

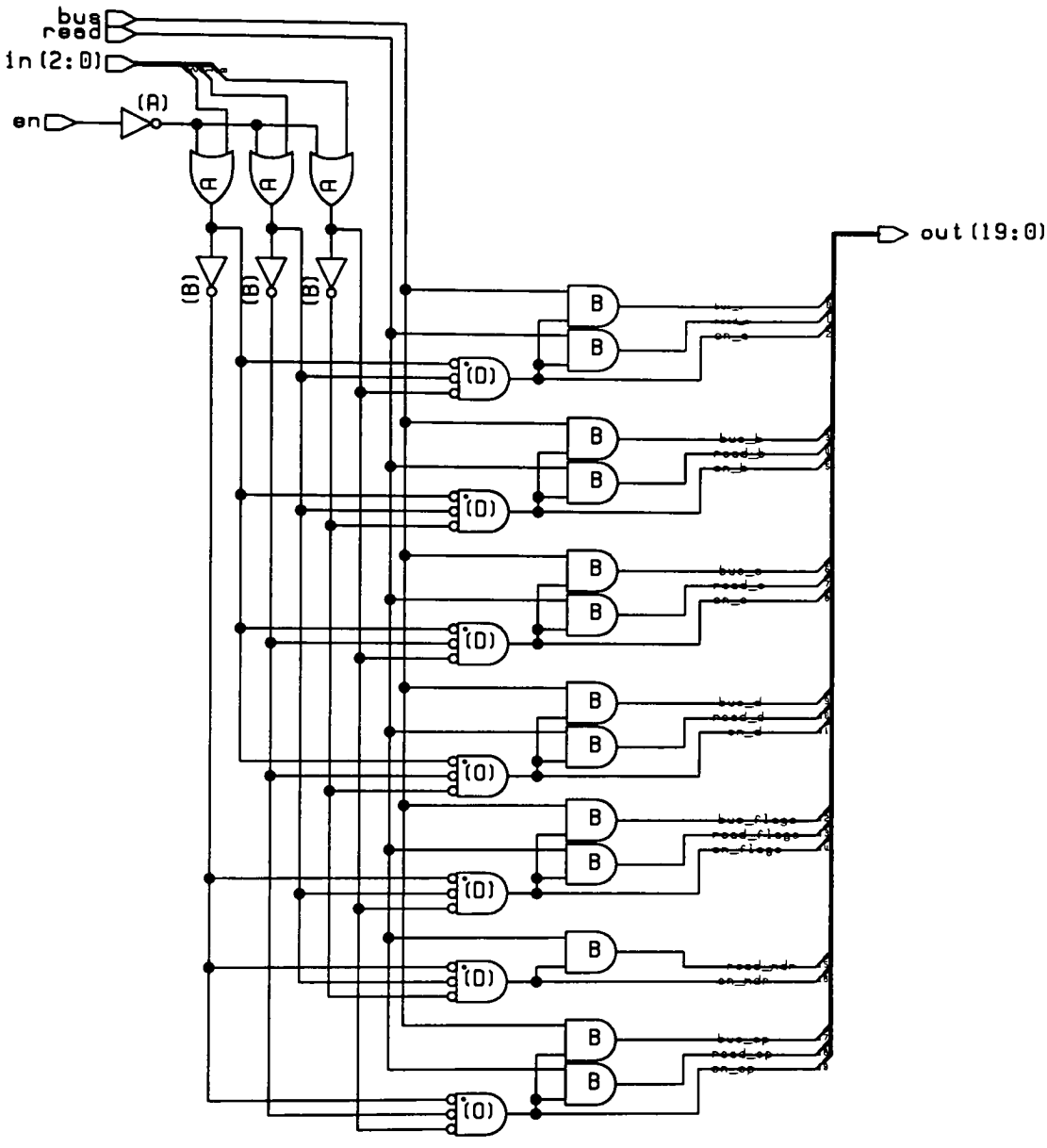

Figure 16 Logic Circuit of Control Unit Register Decode Circuitry

The next block to be discussed is the ALU_CNTRL (ALU Control) block. This logic was necessary due to the fact that the ALU needed to be controlled either through an op-code such as AND, or by the Control Unit explicitly for such things as a CMP which actually subtracts one value from another.

The output labeled *compare* checks to see if the instruction being processed is the CMP instruction. If it is, the Control Unit needs some indication so that it does not try to write the result of the subtraction into any register other than the flags register. The one AND gate used to perform this function could have been placed in the IR and another output from the IR to the Control Unit could have been added. There is no advantage to placing it in one place versus the other.

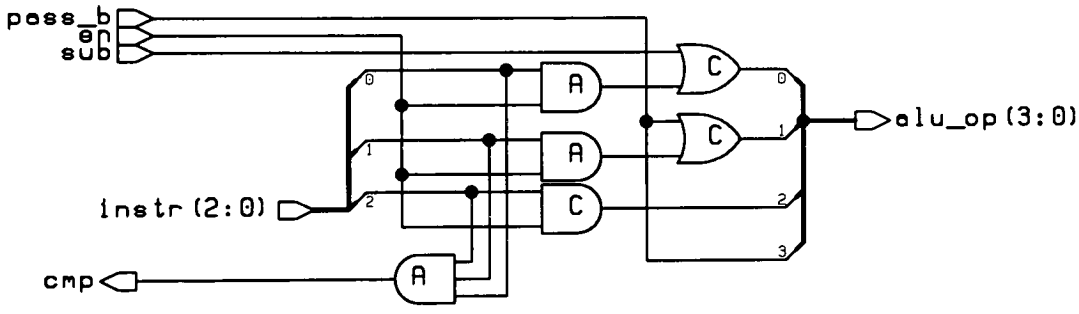

Figure 17 Logic Circuit for ALU Control

The last block of circuitry in the Control Unit is the JUMP_CNTRL (Jump Control) logic. This circuit compares the type of conditional jump that is being processed to the data received from the Flags Register (the lower nibble of bus i0). If there is a match between these two values, a high *jump_good* signal is passed into the POS_PLA where

the appropriate action is taken.

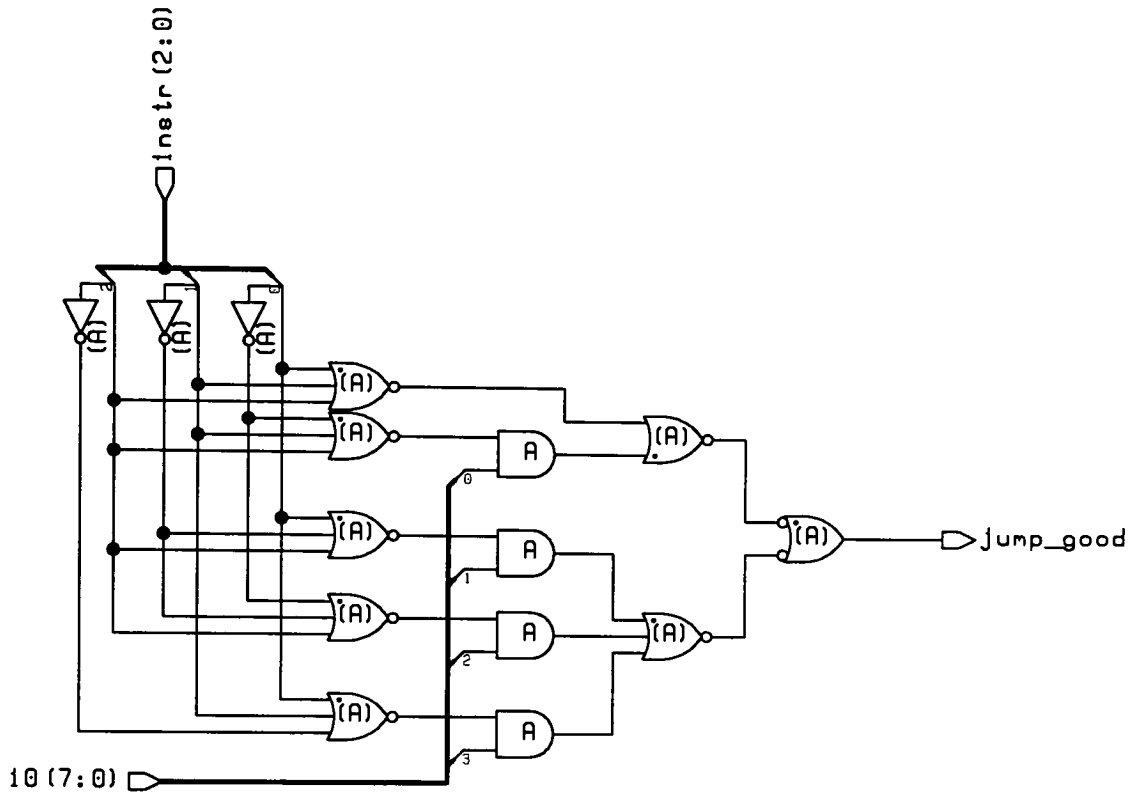

Figure 18 Logic Circuit for Jump Control

The worst case circuit paths in the PLA are defined as those signal paths with the largest number of transistor gates attached to them. These paths would have the largest load capacitance and therefore react the slowest to input stimuli. After simulation of these circuit paths in the Control Unit PLA, the propagation time was measured to be 63 nS.

2.2 Instruction Register

Although this register is discussed before the General Purpose Register in the next section, it was that register that served as the basis for this design.

The circuit for the Instruction Register can be found in Figure 19. On the left portion of the diagram, one can observe the storage portion of the register (a more detailed explanation of circuit operation can be found in the following section 2.3). The instruction decode portion of the circuit demands the most attention. Because of the way in which the instructions were encoded, the most effort (and circuitry) went into the decoding of the 'special' functions such as: Pop, Push, Jump, etc. The remaining circuitry is there simply to decode the addressing mode, general category of the instruction (alu_op, shift_op, nop) and whether the instruction is illegal or not.

An output listing from the Quicksim simulation of the Instruction Register can be found in Appendix D.

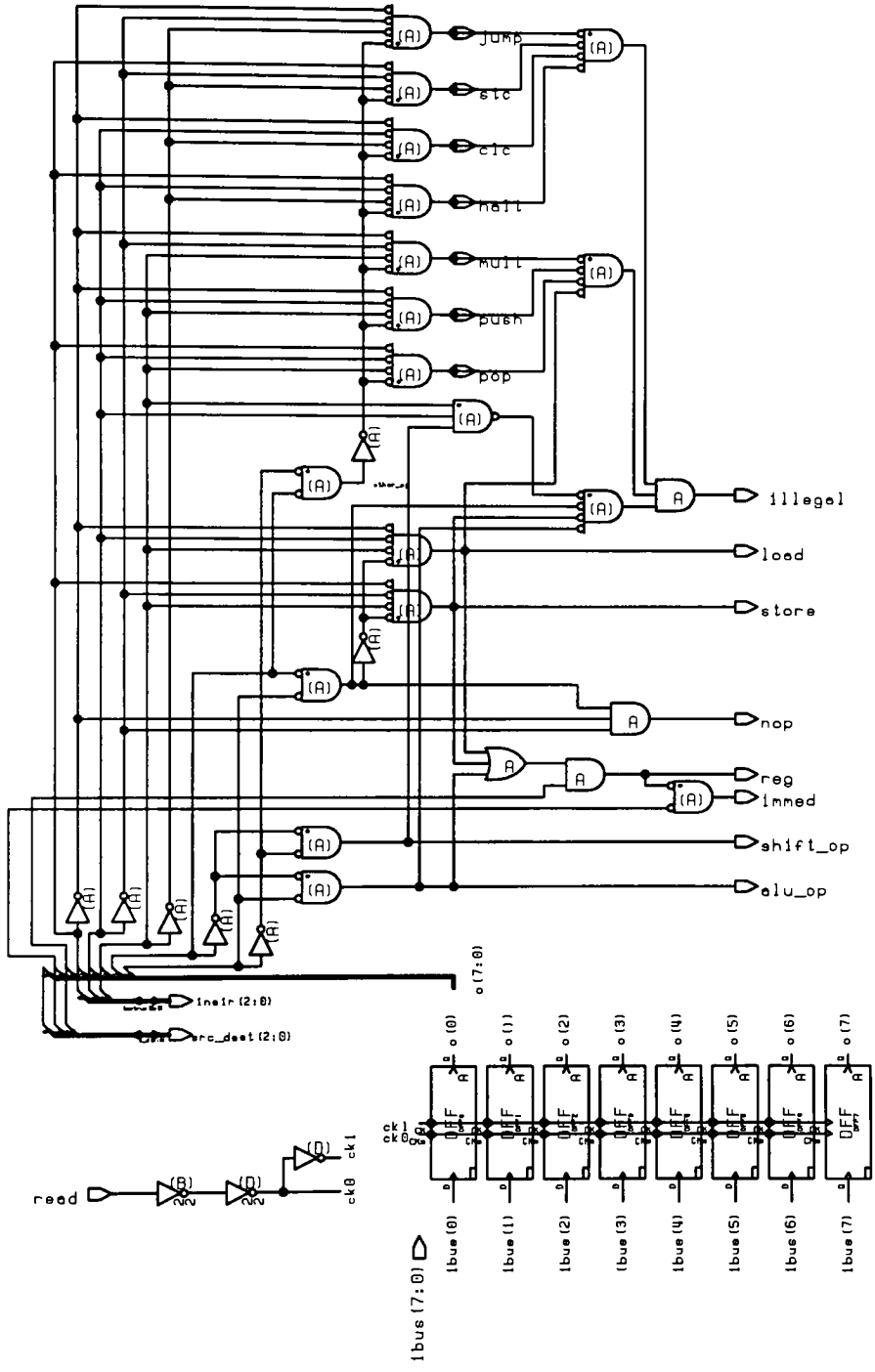

Figure 19 Logic Circuit of Instruction Register

2.3 General Purpose Registers

The general register logic design was made a straightforward exercise by using the d flip-flops blocks designated DFF in The Library. The logic diagram of the register can be seen in Figure 20 below.

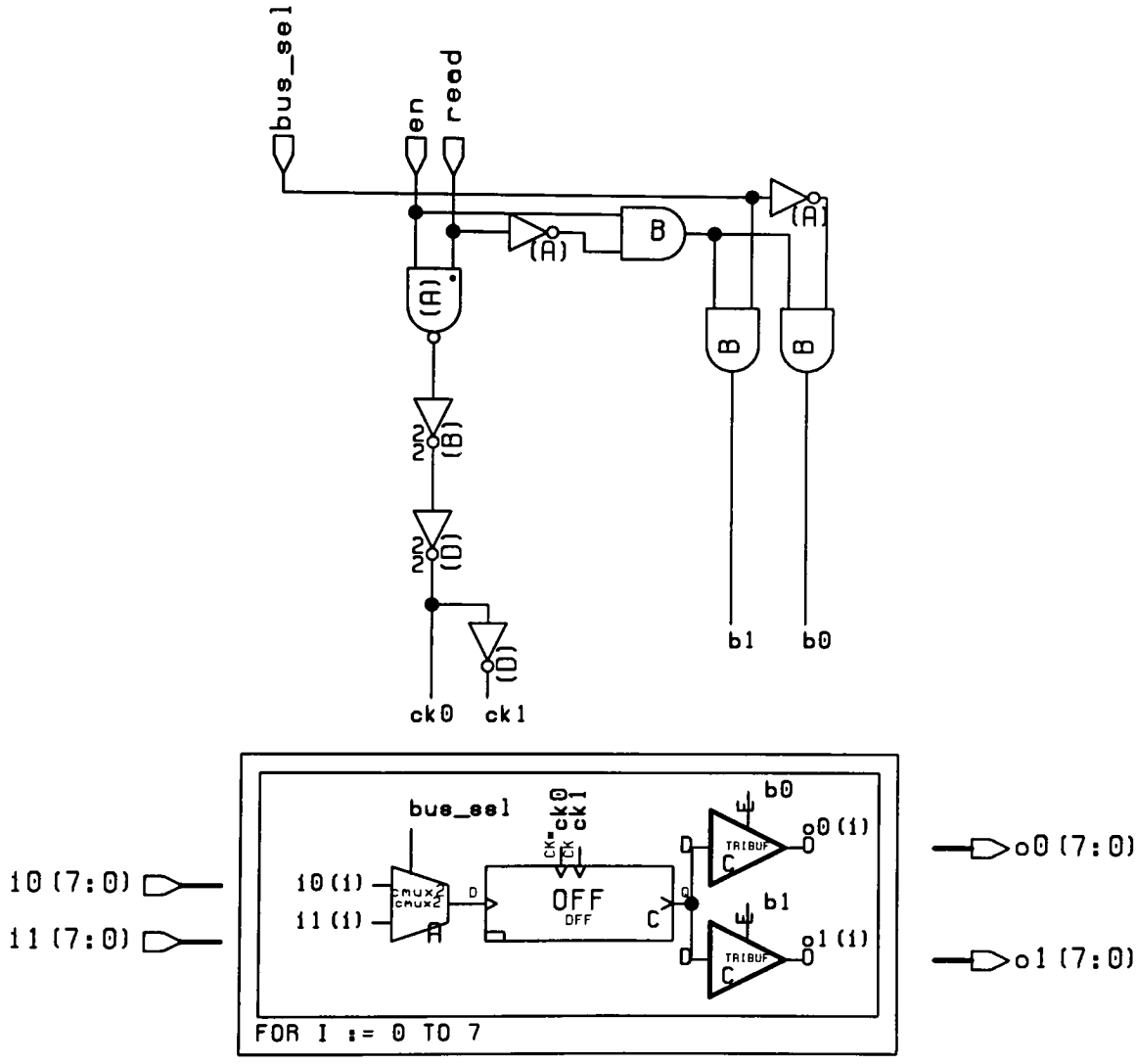

Figure 20 Logic Circuit of General Purpose Registers

The basic operation of the register is simple, the external logic is configured such that the DFF is always accepting data into its master stage. The DFF will latch the data into the slave stage when a high signal is placed in the *read* and *en* inputs. The input *bus_sel* merely causes the input multiplexer to select which of the two busses data is accepted from. The *bus_sel* input also causes one of the tri-state output buffers to become active when the *read* signal is low and the *en* signal is high.

The basic design used in this register will serve as the basis for the design of later registers such as the Stack Pointer Register and the Flags Register.

Figure 21 below is a plot of the logical simulation results for the general register. By examining the diagram, one can observe that the inputs of the register *i0* and *i1* are presented with $AA and $CC respectively. When the *en* and *read* signals are both activated at time 10.0, the register captures the data present at its *i0* input since the *bus_sel* line is low. This data is then written out to the *o0* output at time 39.0. Similarly the value of $CC is read in from *i1* and written to *o1*.

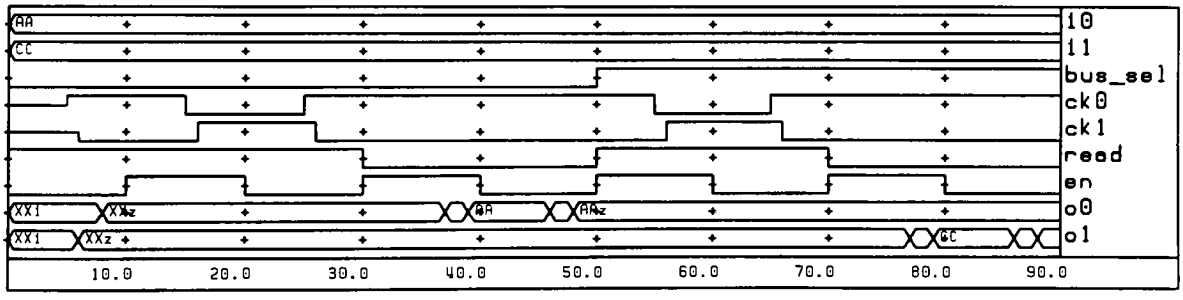

Figure 21 Logic Simulation Results of General Register

The general purpose register was also tested using the Accusim simulator to obtain the time delay of stable signals appearing on the register's output when a write occurs. The signal rise time was 3.56 nS and the fall time was 4.05 nS.

2.4 Stack Pointer

Probably the most difficult register to implement in this processor was the Stack Pointer. The complication of design arises from its need to be self-incrementing and self-decrementing. Much time was spent on this design and several different schemes were tried. The final choice was the simplest conceptually and the most straightforward to implement. By using a bank of adders and asychronously setable and resetable flip-flops it was possible to perform the functions at a reasonable speed with a tolerable circuit size.

The basic register design came from the General Purpose Register. However, due to the extra functionality required by this register, many additions were made. Upon examination of Figure 22 one can pick out the basic circuitry ported from the General

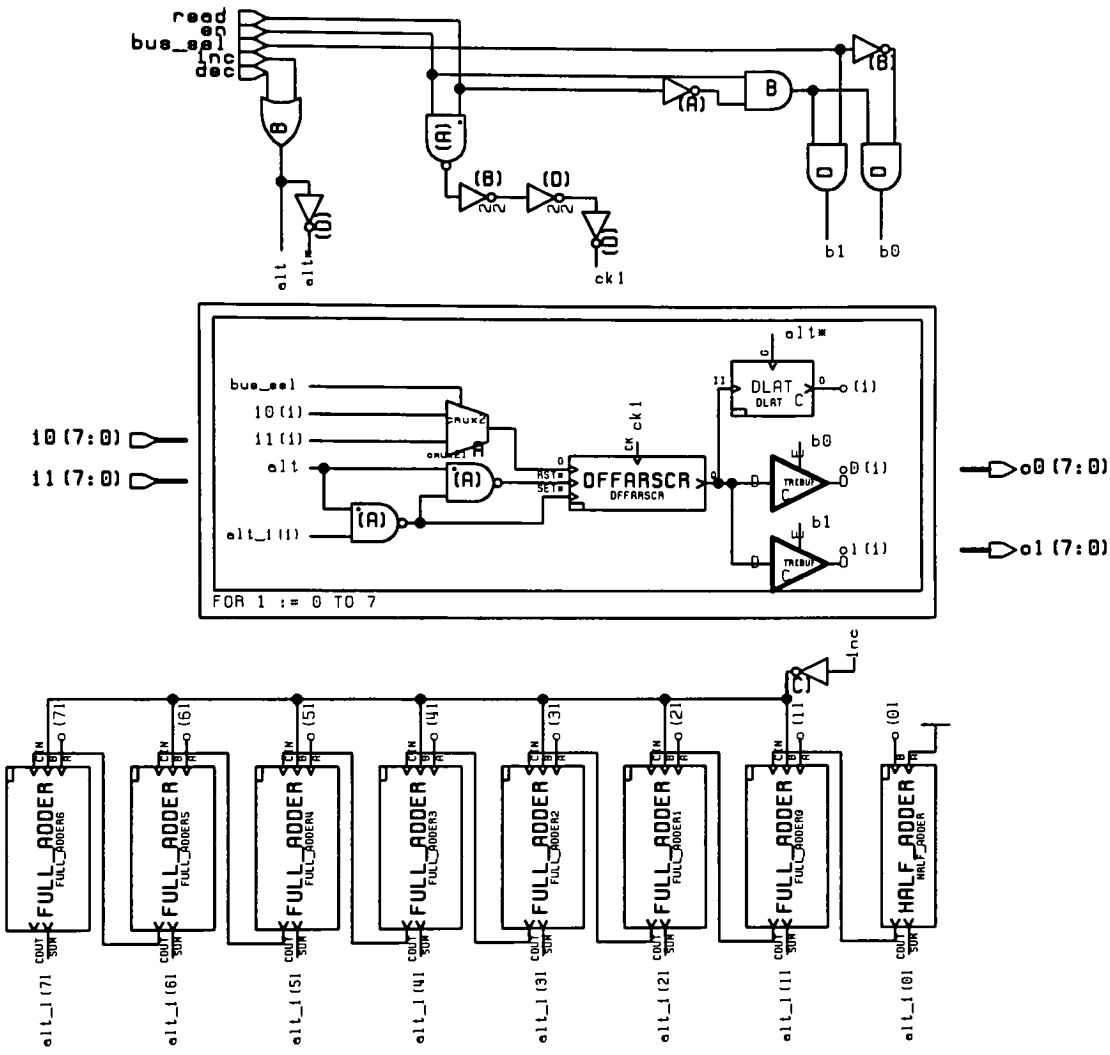

Figure 22 Logic Circuit of Stack Pointer Register

Purpose Register schematic. The way in which the self-incrementing/decrementing is performed is as follows:

1. When either the *inc* or *dec* input is raised the DLAT (d-type latch) circuit captures the present value of the register. The output of the DLAT is

fed into the appropriate bit position in the adder bank.

2. The *inc* control line is used to determine if number added to the present value of the register is 01 or FF, which would be increment or decrement respectively.

3. The two NAND gates at the *rst** and *set** inputs of the DFFARSCR flip-flop is used so that the output of the respective adder block is used to set the flip-flop (if the adder output is 1) or reset the flip-flop (if the adder output is 0).

Since the circuitry contained in this register is very similar to that of the General Purpose Register, it shares the rise and fall time of that register as well. The times required for incrementing and decrementing the values stored in the Stack Pointer were found to be 5.6 nS for both operations. The time was the result of incrementing $FF by 1 and decrementing $00 by 1. These operations caused a carry bit to be propagated throughout the entire adder circuit, thus making it the longest path.

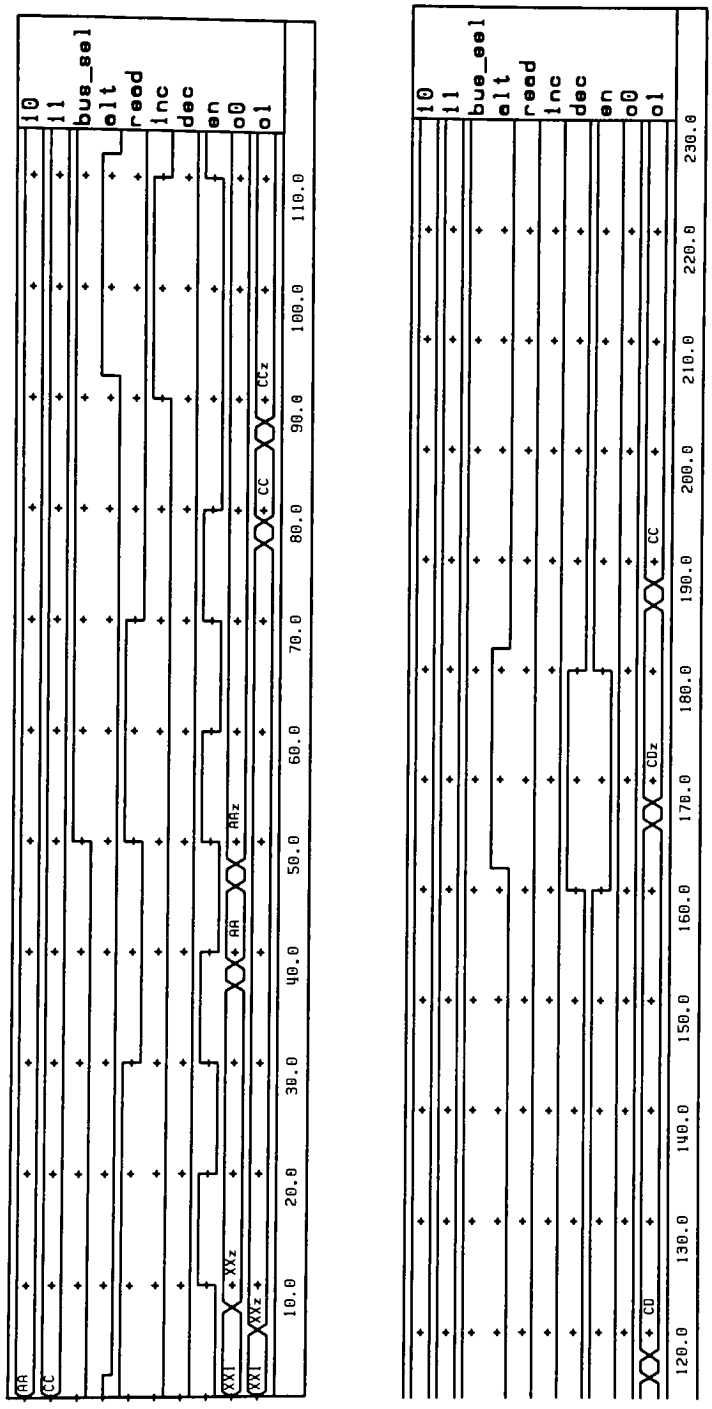

Figure 23 Results of Stack Pointer Simulation

2.5 Flags Register

All registers presented thusfar have been eight bits wide. The Flags Register is the exception. Because there are only four PSW bits, it was not necessary to build a byte register. The previous requirement of 2 output busses was also not implemented here, since the PSW needs only to pass through the ALU or go to the BU to be written to memory. A more detailed explanation of the general design considerations of the Flags Register can be found in Section 1.1.5.

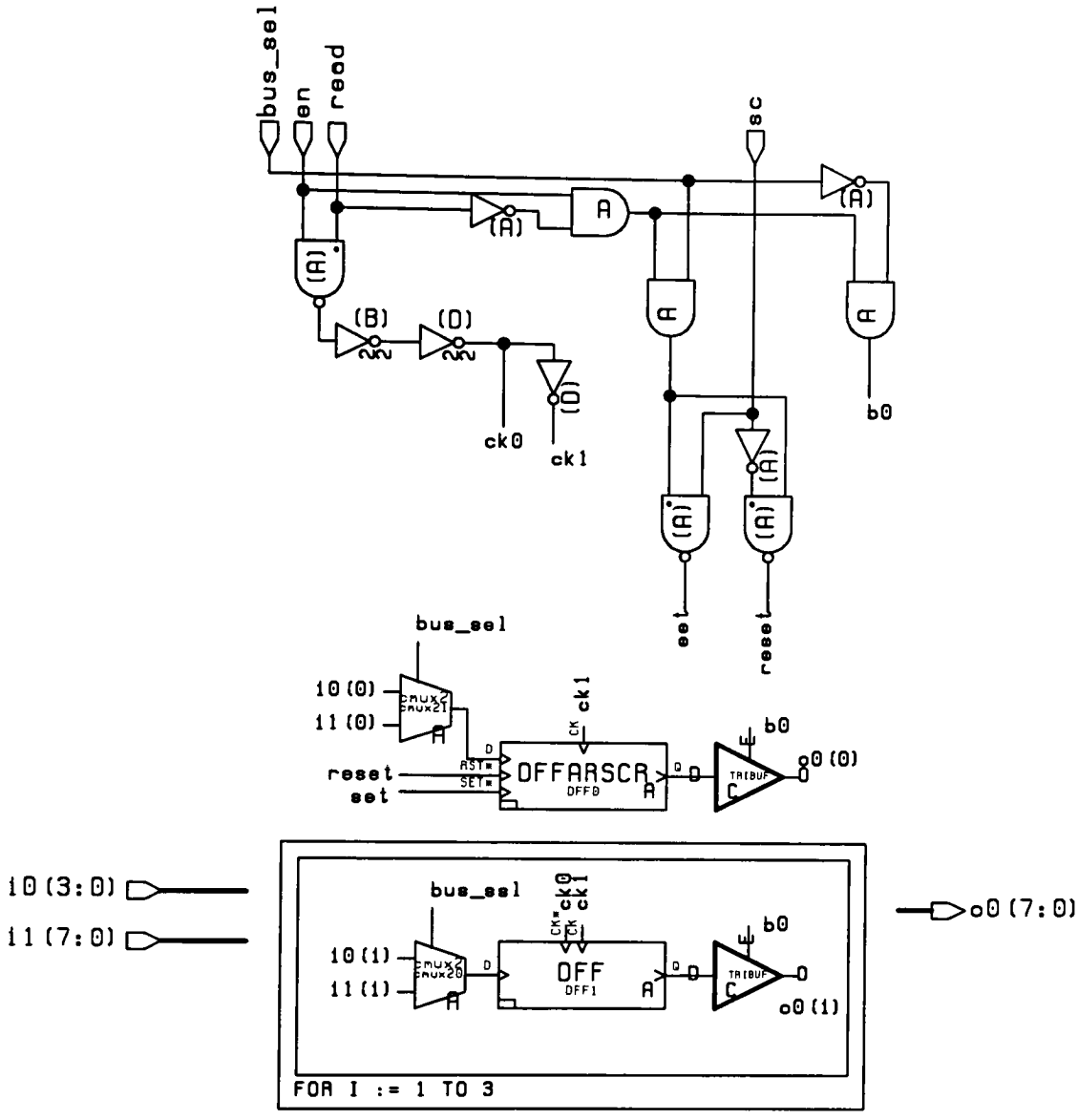

Figure 24 Logic Circuit of Flags Register

The logic to implement the setting and resetting of the carry bit was a simple matter of decoding the *bus_sel*, *read*, *sc* and *en* signals so that when a write to bus1 was attempted the register holding the carry bit would take on the value of the *sc* line.

A test trace of the Flags Register is found below in Figure 25. As once can see from the *o0* output, the value stored in the register is 0x08. The value then changes to 0x09 when a write to bus1 is attempted and *sc* is high. Similarly the stored value returns to 0x08 when *sc* is low and another write is performed.

This register is also a virtual clone of the General Purpose Register. Therefore a physical simulation will not be presented here and the delay times of this register are considered to be the same as those of its origin register.

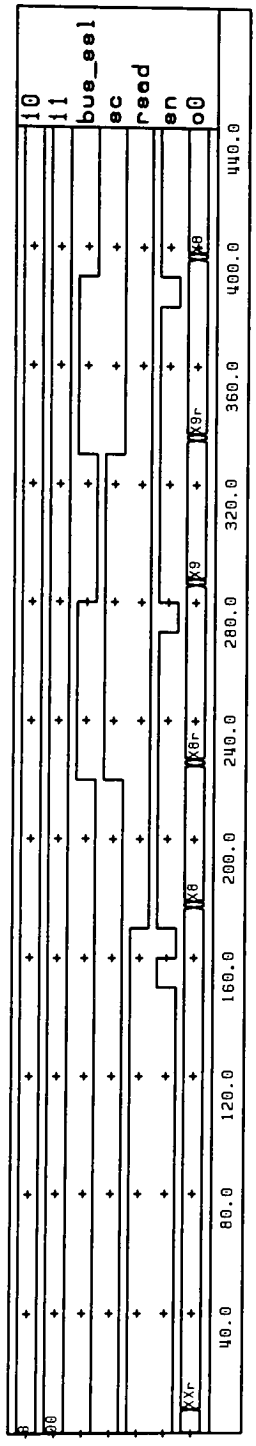

Figure 25 Simulation Results of Flags Register

2.6 ALU

The ALU was one of the most interesting circuits in the processor to design. The ALU is based on the Full-Adder circuit. It was noticed that the Full-Adder truth table contained several logic functions embedded in it. By examining Table 9 it is noticed that if the C variable is maintained at 0, then the Sum column is actually B A, the Carry column is B·A. And if C is made a 1 then the Sum and Carry columns become $\overline{B\ A}$ and B+A respectively.

C	BA	Sum	Carry
0	00	0	0
0	01	1	0
0	10	1	0
0	11	0	1
1	00	1	0
1	01	0	1
1	10	0	1
1	11	1	1

TABLE 9 ALU Truth Table

By making these observations, it was apparent that four of the nine functions ($\overline{B\ A}$ was not used and addition is the function of the Full-Adder) were already available for use. A scheme was developed using extra logic, to create the remaining functions. The following table summarizes the remaining functions to be implemented and the technique in which it was done.

NEG B: B = $\overline{B}$; A = 0, Carry-in = 1
NOT B: B = $\overline{B}$; A = 0
PASS_A: B = 0 ; A = A
PASS_B: A = 0 ; B = B
SUB: B = $\overline{B}$; A = A

Signals that needed to be inverted for use in the full adder were passed through Exclusive-Or gates which served as programmable inverters. At certain times, a zero was required at one of the inputs to the adder. This was accomplished by AND gates placed before the Exclusive-Or gates which could be made to pass a 0 simply by placing a low signal on one of its inputs.

The problem of choosing between the Sum and Carry column functions was easily solved through the use of multiplexers at the output. These multipliexers were controlled by the decode logic to select the proper adder outputs.

The last circuitry to be implemented was that which generated the PSW bits. The **C**arry bit (when applicable) was simply the Carry-out of the most significant Full-adder block. The Sum output of the most significant stage was passed as the **N**egative bit. The zero signal can be described by the equation: **Z**ero = $\overline{a7}\cdot\overline{a6}\cdot\ldots\cdot\overline{a0}$. Finally the **O**verflow could only be true during an add when the most significant bit (MSb) of the *a* and *b* inputs differed from the MSb of the output, or **O** = $\overline{out(7)}\cdot a(7)\cdot b(7) + out(7)\cdot\overline{a(7)}\cdot\overline{b(7)}$.

Physical simulations of the ALU through its longest circuit paths were performed to calculate the maximum time required for the output lines to settle at correct values. The traces of the simulations are too large to be seen clearly when printed so they will be excluded. The maximum circuit path involved an add with a carry that propagated though the entire circuit. This can be realized by adding $FF with $01. Since the adder design is essentially a ripple adder, the carry must pass through the entire circuit before the carry bit of the PSW will become valid. The low to high signal delay time for all of the outputs was measured to be 9.77 nS and the high to low delay 53time was 8.11 nS.

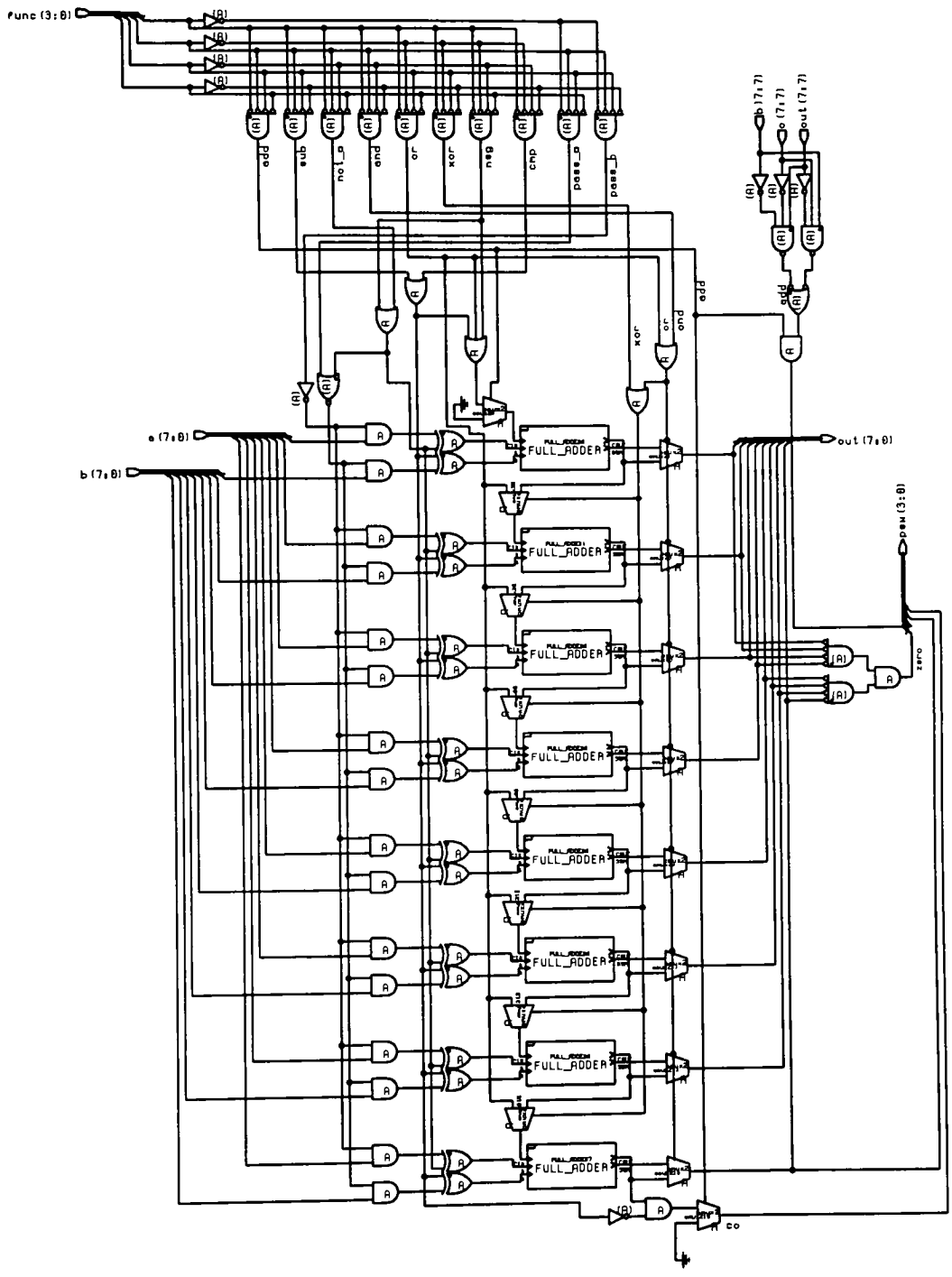

Figure 26 Logic Circuit of ALU

2.6 Multiplier

The next block to be discussed is the multiplier block. Figure 27 is a schematic of the highest level of hierarchy in the multiplier. The core has 16 inputs (2 eight-bit busses) and a 16 bit output broken into two bytes *out_hi* and *out_lo*. The results of a multiplication appear at the outputs and are permitted out of the circuit when the *oe* signal is raised enabling the tri-state buffers.

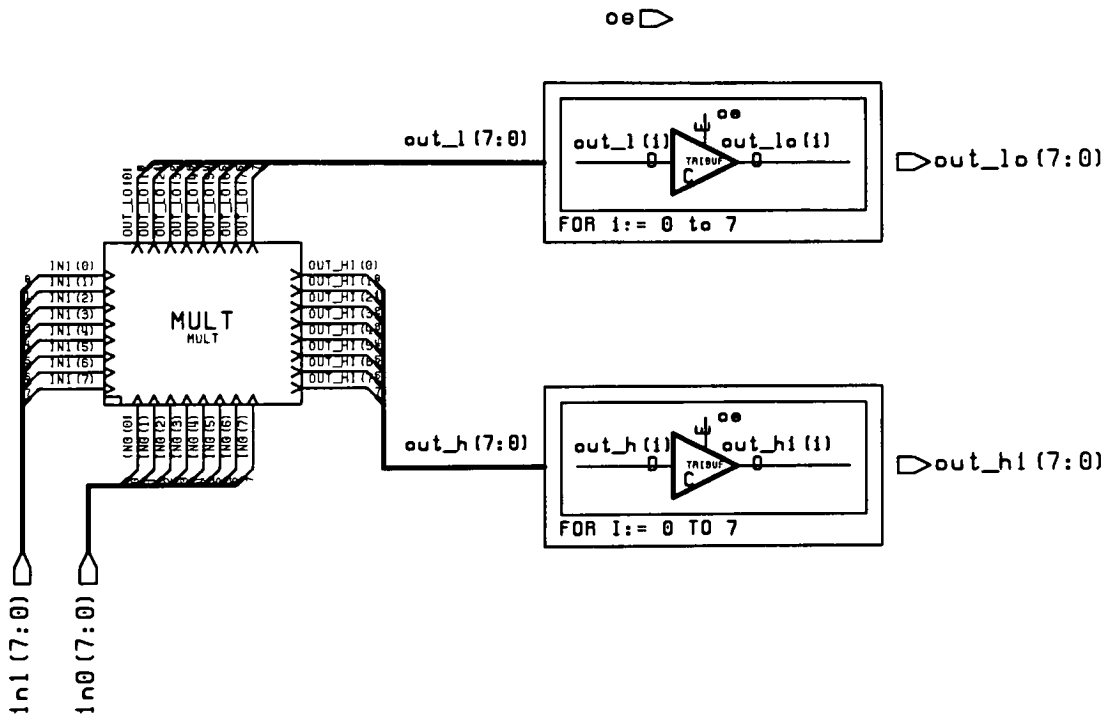

Figure 27 Logic Circuit of Highest Multiplier Hierarchy Level

The multiplier core (mult) is shown in Figure 28 as a combinational circuit made up of mainly full adder blocks and NAND gates. The algorithm for multiplication is

based on a Modification of Booth's Algorithm.

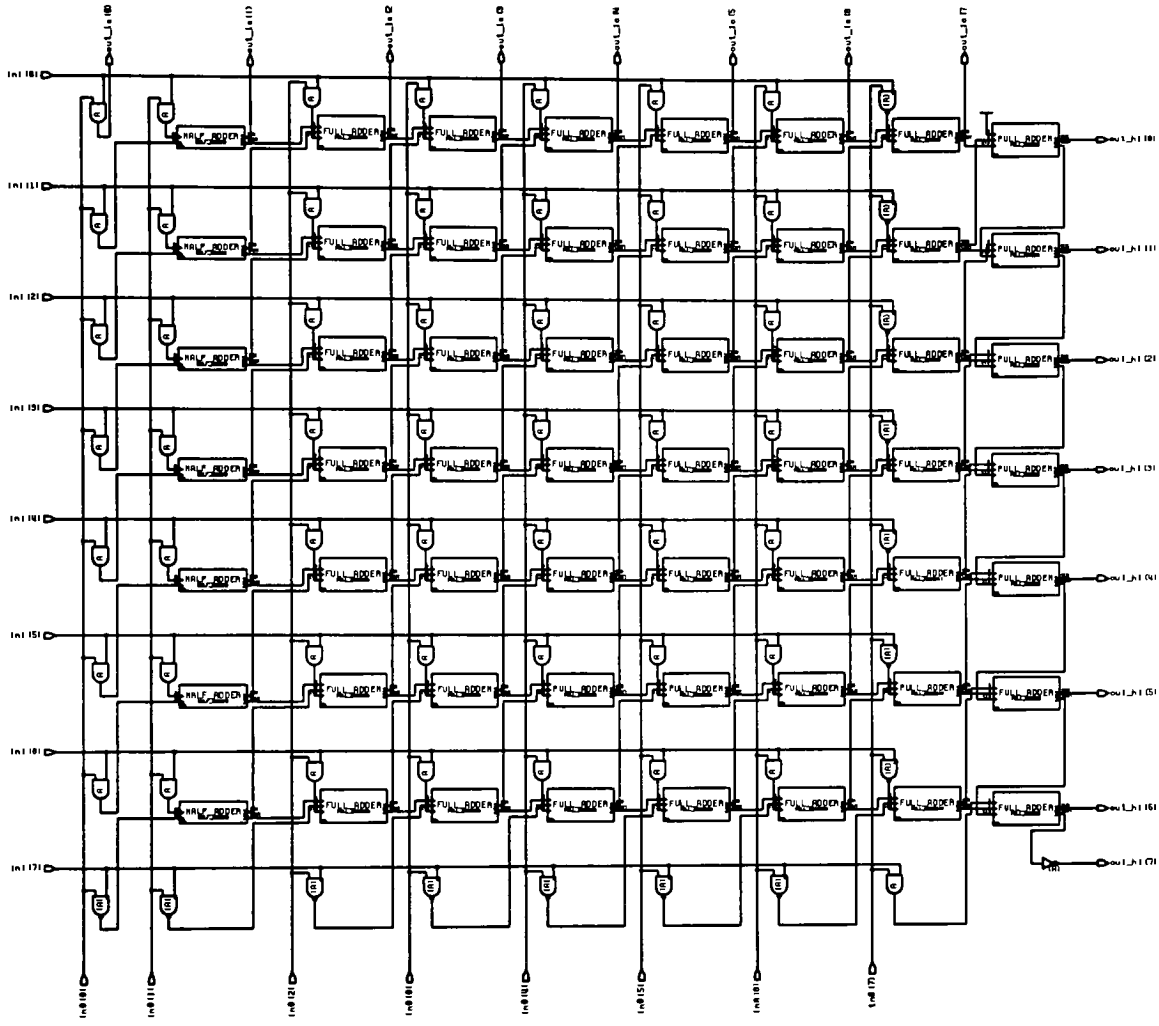

Figure 28 Logic Circuit of Multiplier Core

Figure 29 is an example of a few numbers passed into the multiplier during a test run.

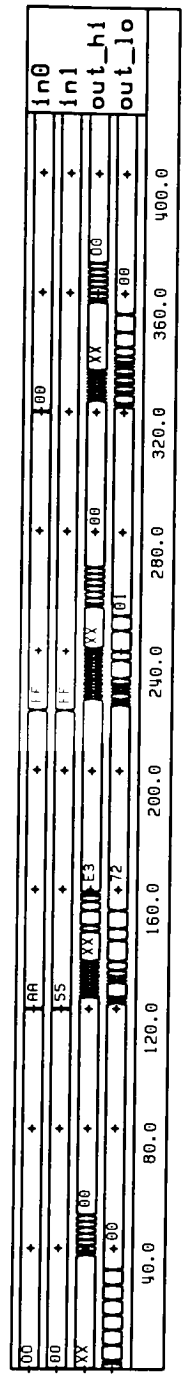

Figure 29 Simulation Results of Multiplier

Because of the complexity of the multiplier and its presence in the data path, it was expected that this block would be the limiting factor in the speed of processor operation. By examining the circuit, it is possible to deduce that the longest circuit path that is used in the multiplication of $FF by $FF. Using Accusim, the time for all signals to become stable with this multiplication was found to be 14.89 nS.

2.8 Memory Data Register

The register that was by far the easiest to implement is the Memory Data Register. The circuitry is the same as that of the General Purpose Register except for its simplicity. By examining Figure 30 it can be see that the logic for the register is straighforward because of the existance of only one input and one output bus. The only new additions to the basic design were the high and low nibble busses which required no extra logic.

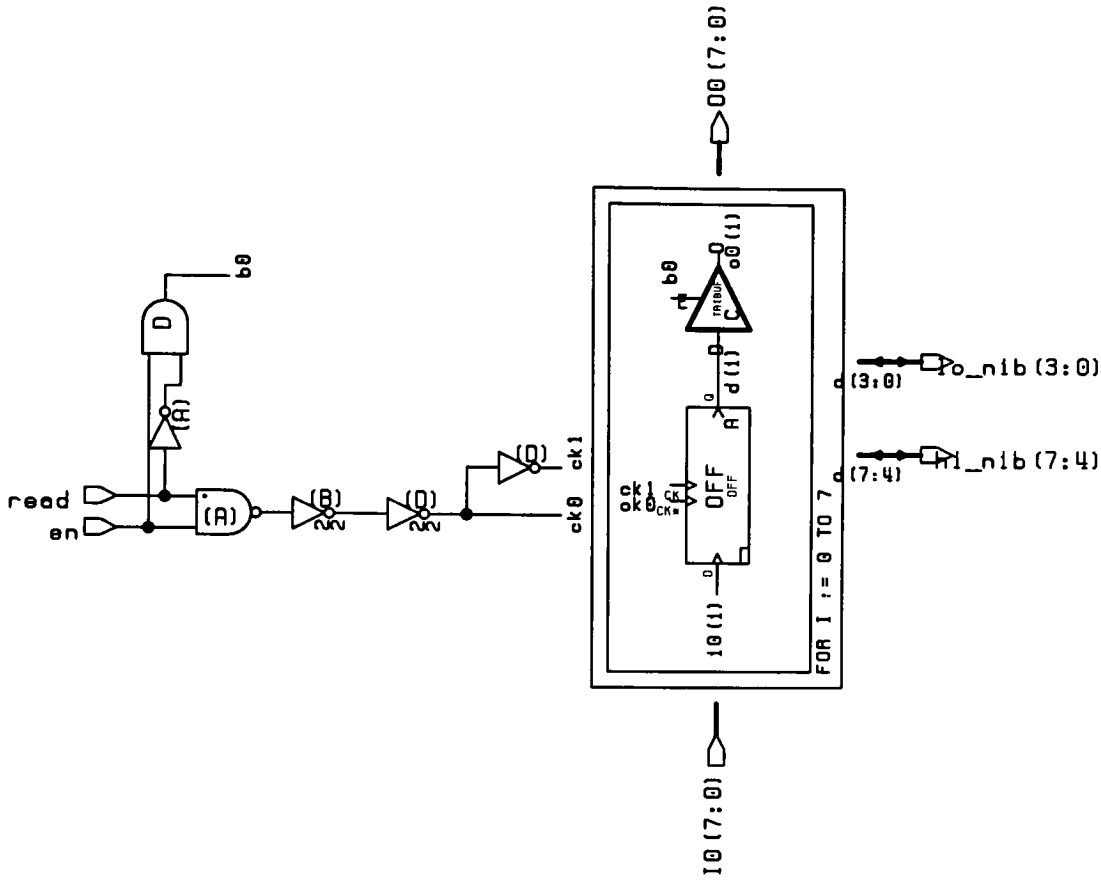

Figure 30 Logic Circuit of Memory Data Register

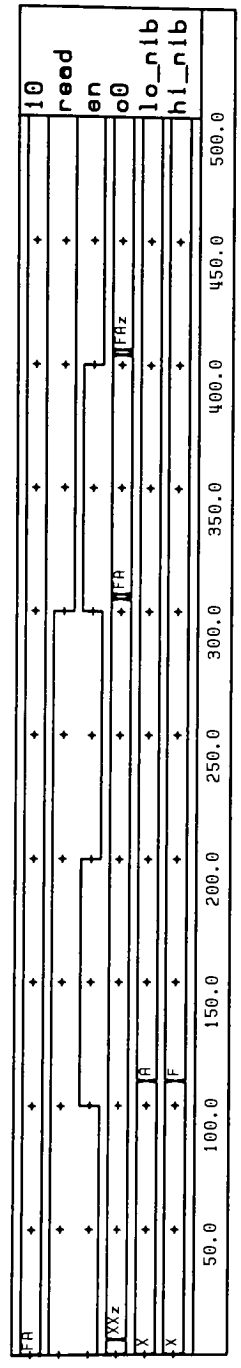

Figure 31 Simulation Results of Memory Data Register

2.9 Shifter

The construction of the shifter circuit was simple. Since the shifter was to be combinational it could be done by simply switching bit positions within the byte being operated on. The most straightforward way to accomplish this was through the use of multipliexers. This was also the most efficient way to accomplish the shifting due to the fact that the multiplexers were implemented using CMOS transmission gates.

There are two varieties of multiplexers used in the design. The middle six bits use two input muxes to shift either left or right. The most and least significant bits require four bit (actually four and three bits respectively since the LSb does not maintain a sign bit on an ASL or LSL, which are functionally identical) muxes because of the possibility of replacing those bits with the carry-in bit, a zero, the sign bit, or a shifted bit.

There is also logic present to generate the PSW. The **Z**ero bit is evaluated in the same way as it was for the ALU (see Section 2.6). The **O**verflow bit is always set to a low since no overflow is possible in the shifter. The most significant bit is passed out as the **N**egative bit and the carry bit is the original *din(7)* or *din(0)* for a left shift or right shift respectively.

After logical simulation, the longest path in the circuit was simulated with Accusim to obtain the limiting propagation time. The actual propagation time for a stable signal through the shifter was 1.82 nS using an ROR instruction which imports a new carry bit. The time that limits the speed of shifter operation was that required to generate the PSW. The time until the Zero bit became stable was 5.18 nS. This is to be expected since the generation of this bit requires all other bits to be stable.

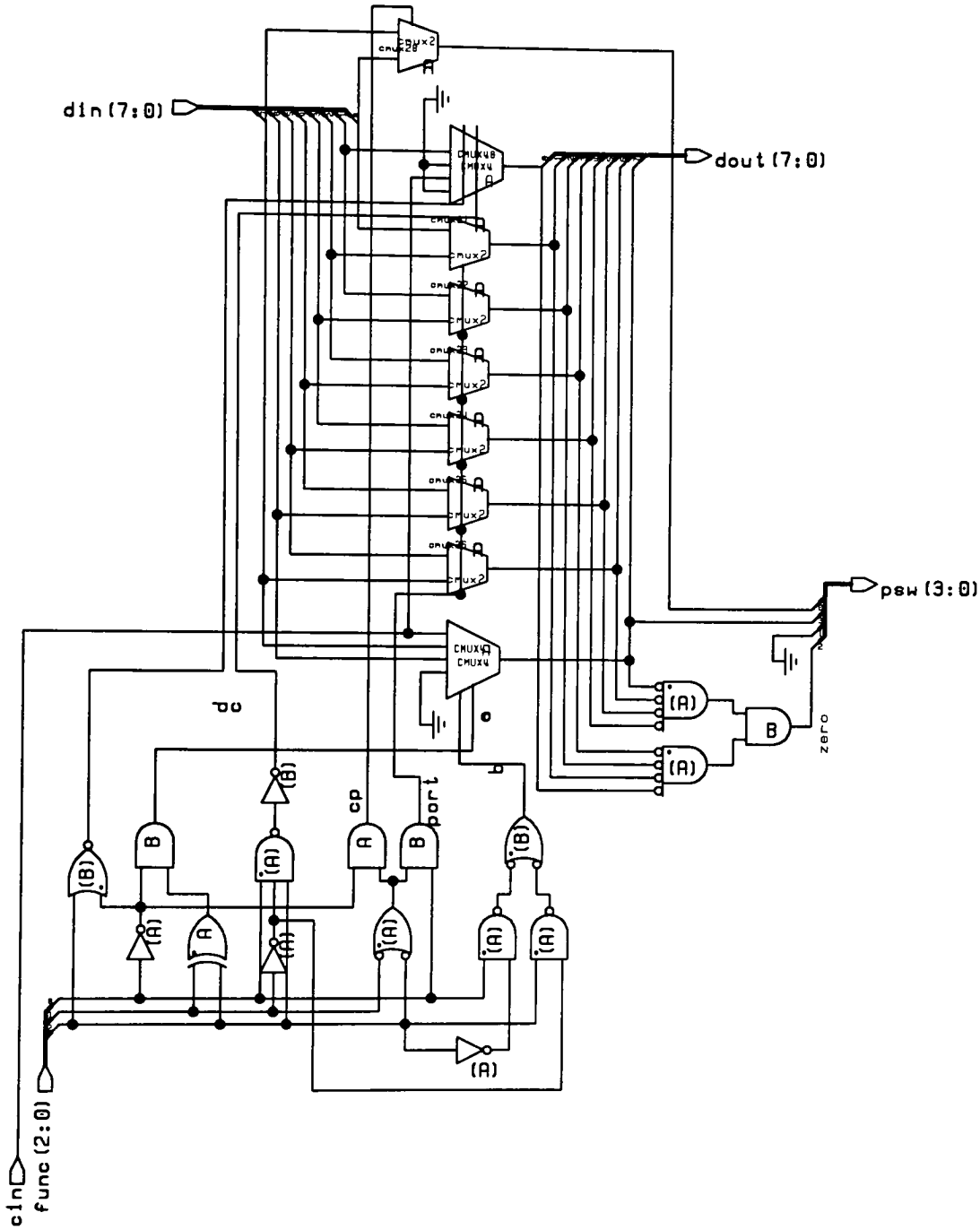

Figure 32 Logic Circuit of Shifter

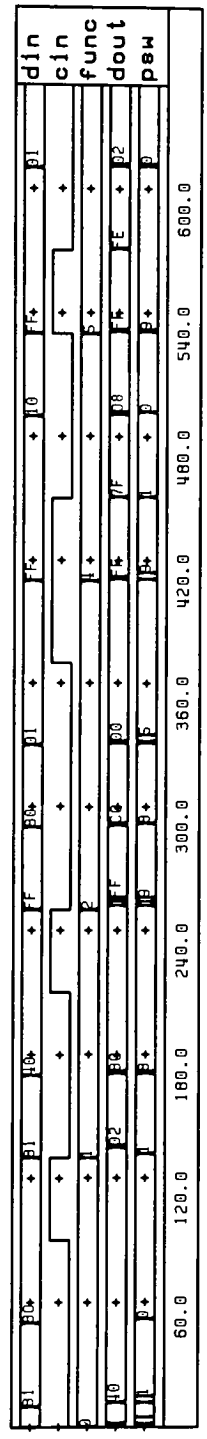

Figure 33 Simulation Results of Shifter

2.10 Bus Unit

The circuit shown in Figure 36 is the top level schematic of the Bus Unit. This block was the most complicated to design of the entire processor. The complexity arose from the many functions that it performs such as reading from and writing to memory, and maintaining the prefetch and the functions involved with the prefetch, like direct fetching.

The PLA and its associated latches is shown in Figure 37. This PLA controls the various other parts of the Bus Unit. The latches are used to capture the outputs of the PLA when they become stable.

The next schematic, Figure 38, is that of the Bus Unit Timer. This timer is used to provide adequate setup up and hold times for writes and reads to and from memory. The important times associated with memory reads and writes are shown in Figures 34 and 35 respectively.

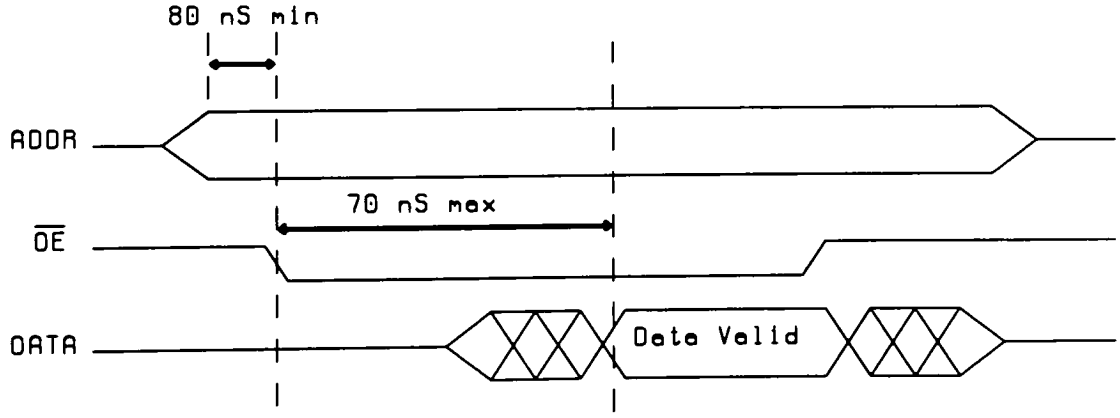

Figure 34 Read Cycle Timing Diagram

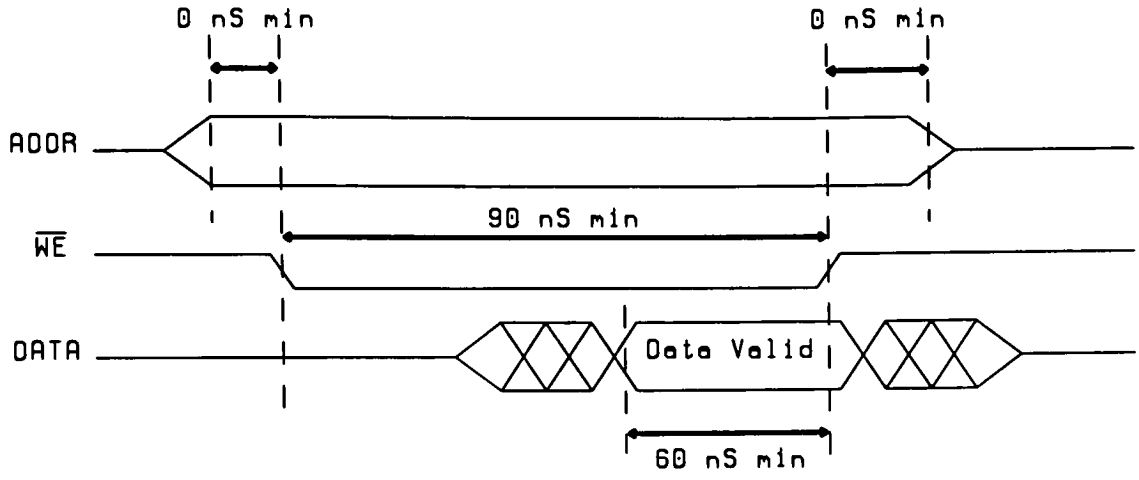

Figure 35 Write Cycle Timing Diagram

The different outputs on the timer correspond to lengths of time that must have passed before certain actions can be taken. For instance in Figure B-3, the Bus Unit would remain in State C until the *setup1* time has elapsed.

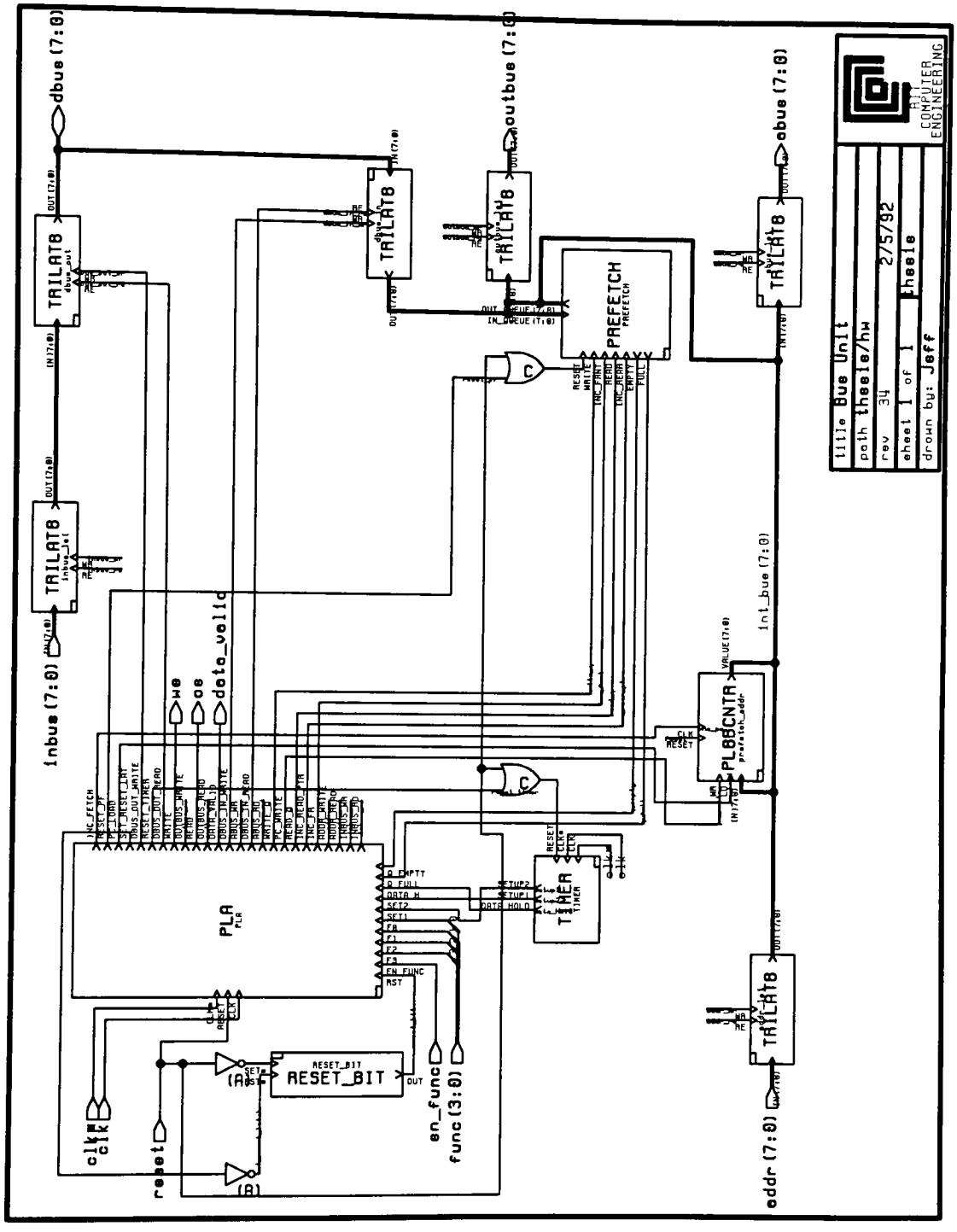

Figure 36 Logic Circuit for Bus Unit

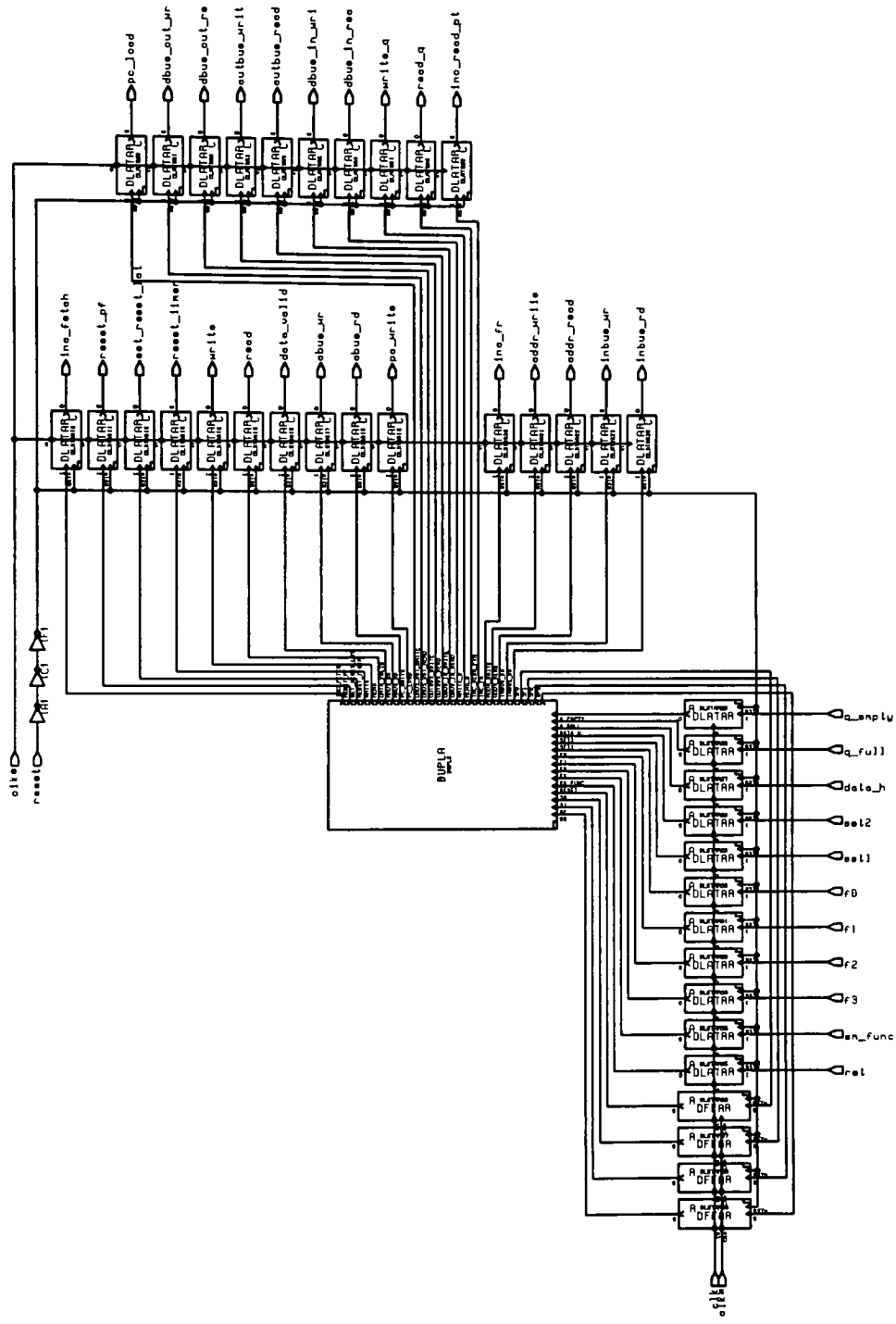

Figure 37 Logic Circuit for Bus Unit PLA

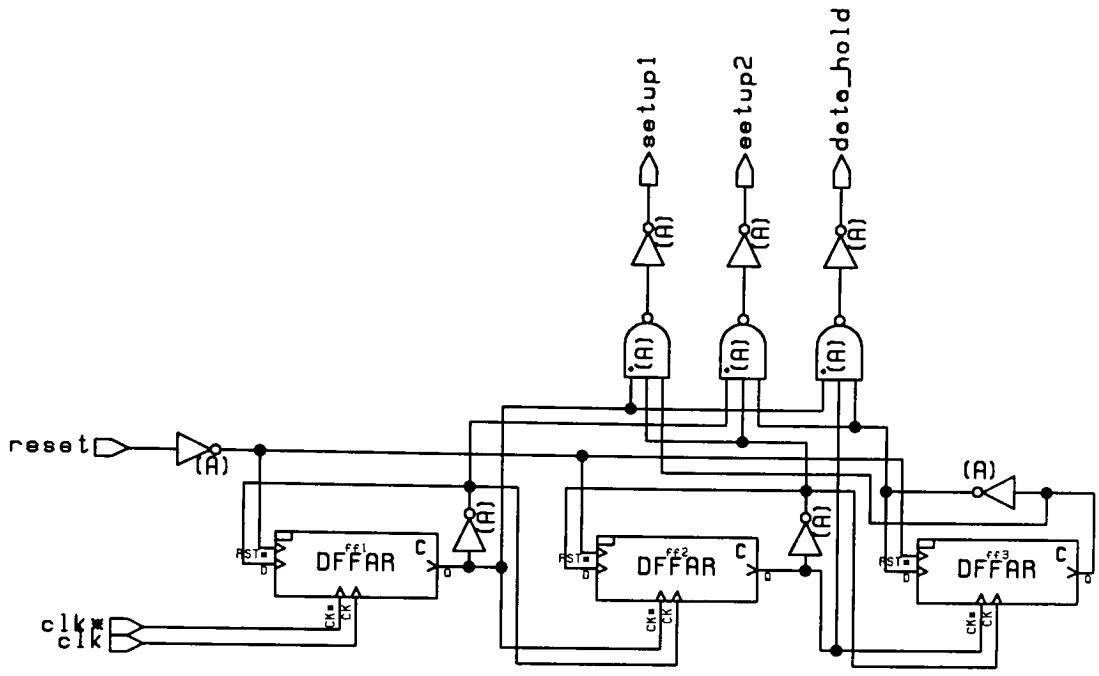

Figure 38 Logic Circuit for Bus Unit Timer

Figure 39 shows the circuit for the parallel loaded eight-bit counter. This counter is used as the PC in the Bus Unit. The counter is a ripple counter that is activated when its *clk* line is pulsed. It also has the capability to reset to all zeros when the *reset* line is brought low. Using the set and reset inputs of the DFFARSCR flip-flop, some external logic is used to selectively reset, or set the flip-flops according to the value of the appropriate input bit. The set value is loaded when the *ld* line is brought high.

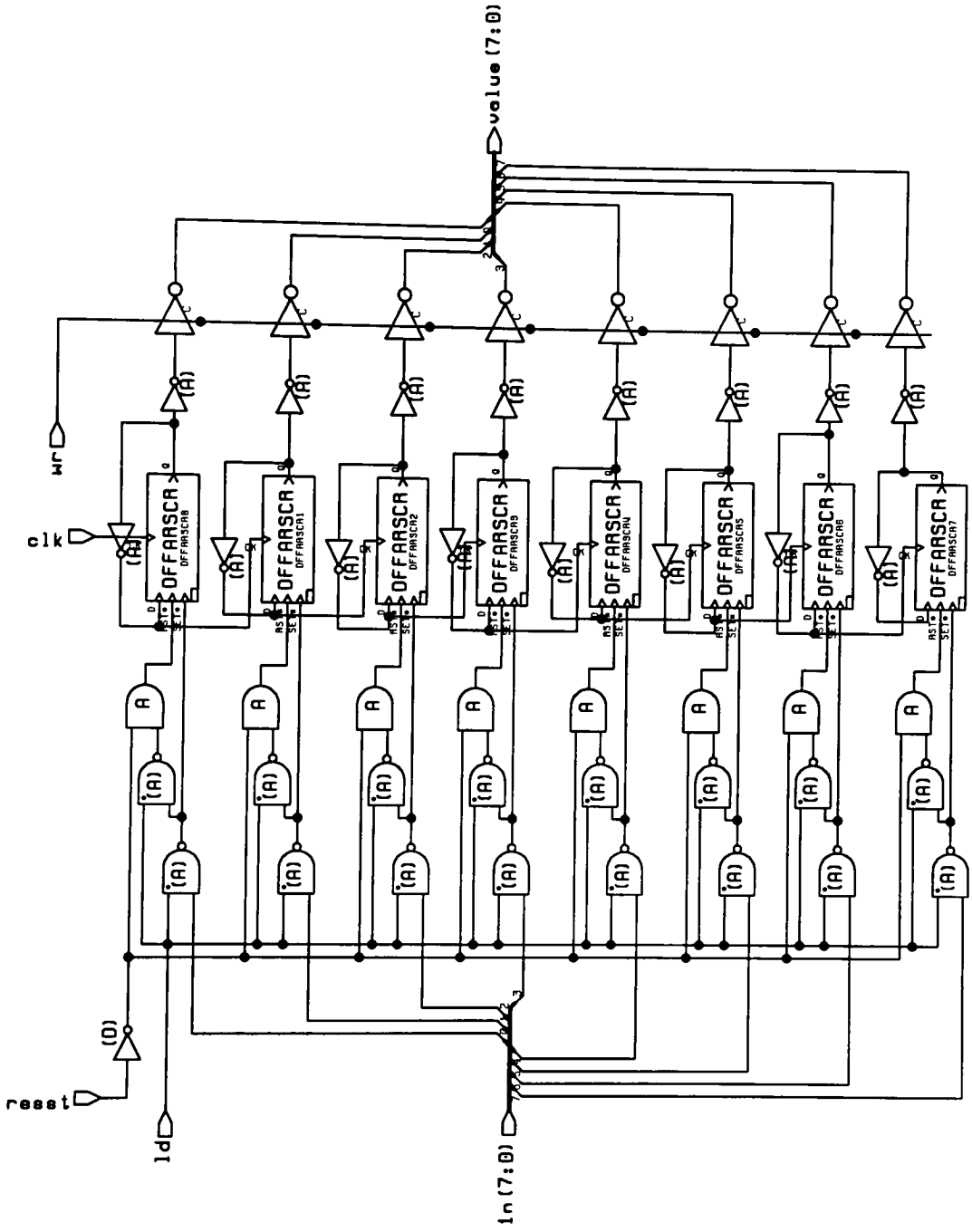

Figure 39 Logic for Parallel Load 8-bit Counter

Although this counter is nearly identical to the Stack Pointer, it was implemented differently for variety and to investigate the difference in performance between a ripple counter and the full-adder implementation used for the stack pointer. As expected, the full adder SP implementation was superior. An increment in the parallel loaded eight-bit register took 27.8 nS through the longest path. This register has no decrement capability.

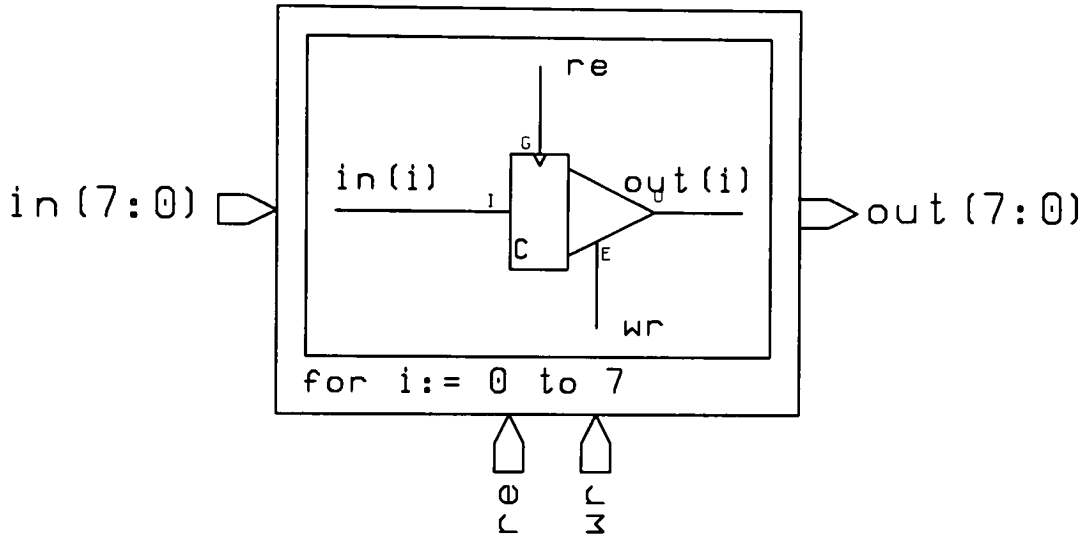

Figure 40 Logic for 8-bit Tri-State Latch

Figure 40 is the schematic representation of the eight bit tristate latch created for the processor. This latch serves to hold data that needs to be written to memory or an internal bus. The tristate capability can be used for shared busses either internal or external to the processor.

In order to create the eight bit latch above, it was necessary to design the tristate latch it uses. Although there were latches in The Library as well as tristate buffers,

there was no part that combined the two, and the tristate buffers that do exist are used to drive extremely large capacitive loads.

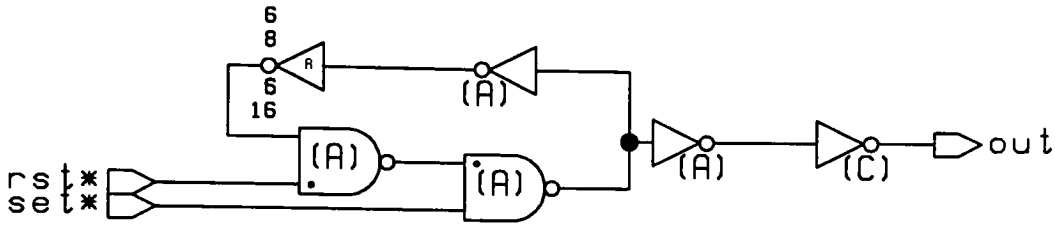

Figure 41 Logic for Reset State Latch

After the processor is reset, the Bus Unit will be place in the default state (all PLA state inputs will be 0) since a reset clears the contents of the PLA's latches. From this state the BU will automatically bootstrap itself by going out to memory location $00 to fetch the starting address of the start of the program code. Since some of the states of the BU that are used after a reset are the same as those used in normal instruction fetching, some sort of 'flag' is needed to inform the PLA of its present condition. The circuit of Figure 41 serves this purpose.

The master reset line is connect to the *set** input of the Reset State Latch. When a reset occurs the output of the latch becomes high and serves to inform the BU that it has been reset. After the BU has performed all of the steps called for after a reset, the BU will clear the latch by toggling the *rst** input of the latch.

During normal operation, the BU attempts to pre-fetch instructions and store them in an internal queue for later access. The circuitry that implements this function is shown as Figure 42. Although the prefetch is actually only four elements deep, the prefetch system uses five storage elements in order to differentiate between the queue

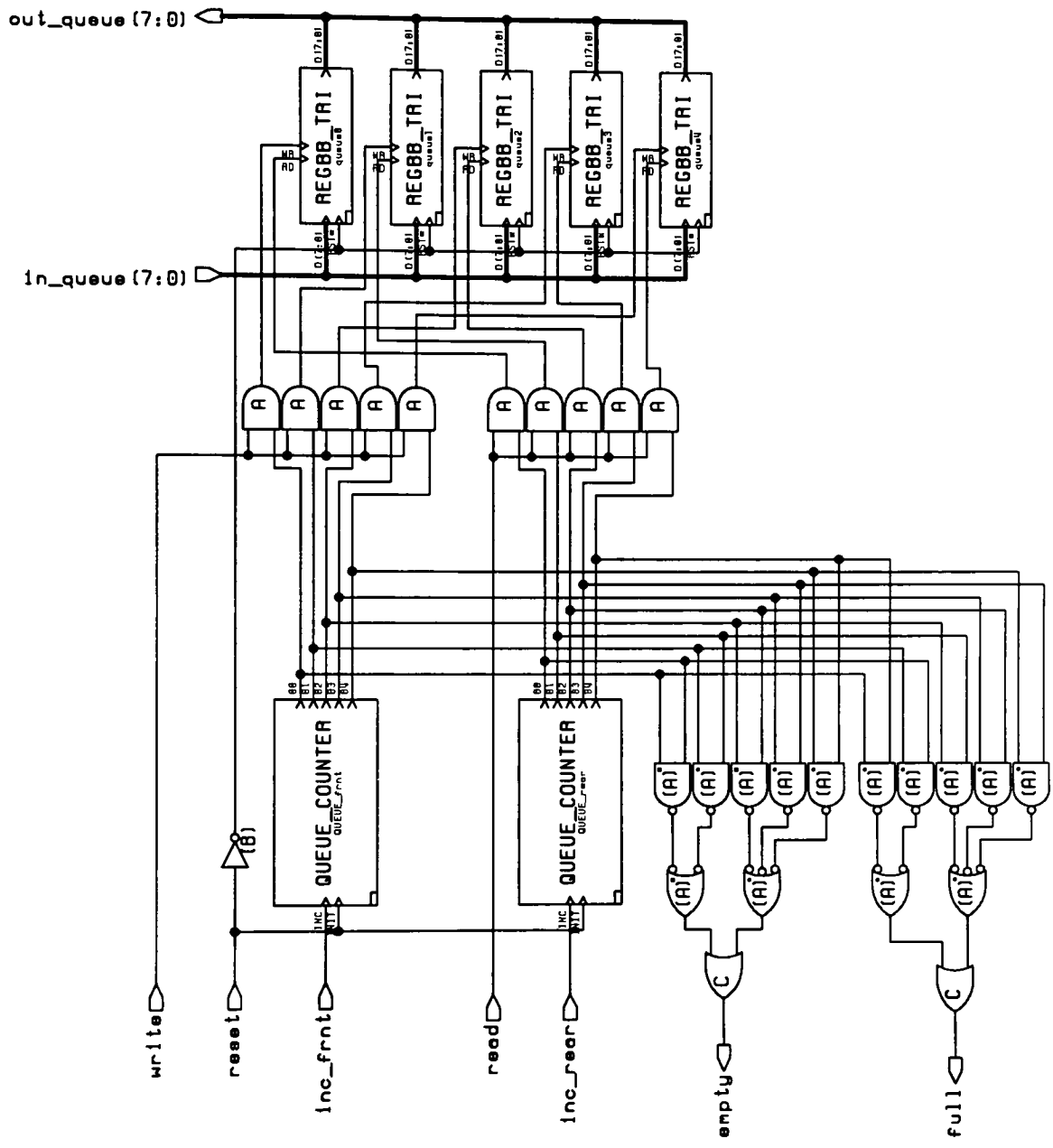

Figure 42 Logic for Prefetch Queue

being full, empty, or in between.

The manner in which the prefetch operates is actually very straightforward. Each of the storages elements designated REG8B_TRI is an 8-bit tri-state register. The read and write lines of these registers are controlled by the outputs of two counters and AND gates to enable or disable the reading and writing. The write input of each register is connect to the QUEUE_frnt counter and the read input of each register is connected to the QUEUE_rear counter.

During normal operation, the rear counter controls which of the registers will accept the new data, whereas the front counter designates which register will present its data to the processor. To enter a new piece of data into the prefetch, the *read* input is raised. This sensitizes the AND gates to pass a high signal to whichever register is to read the data as specified by the rear counter. After the read takes place, the *read* line is lowered and the counter incremented. The operation of making the prefetch write to the processor's internal data bus operates similarly.

Also present in the prefetch are two circuits to determine if the prefetch is full or empty. A full prefetch occurs when the rear counter has been incremented to the point where one more increment would make the front and rear counters. equal. An empty queue occurs if the front and rear counter are equal.

Using two counter circuits called QUEUE_COUNTER in the diagram, the front and rear queue pointers are maintained. The actual implementation of the counter is shown in Figure 43 as a ring-counter. This category of circuit was chosen because no decoding was required to enable a particular storage element. All that was need were AND gates at the output of the counter to control which of the storage elements read or wrote from and to the bus.

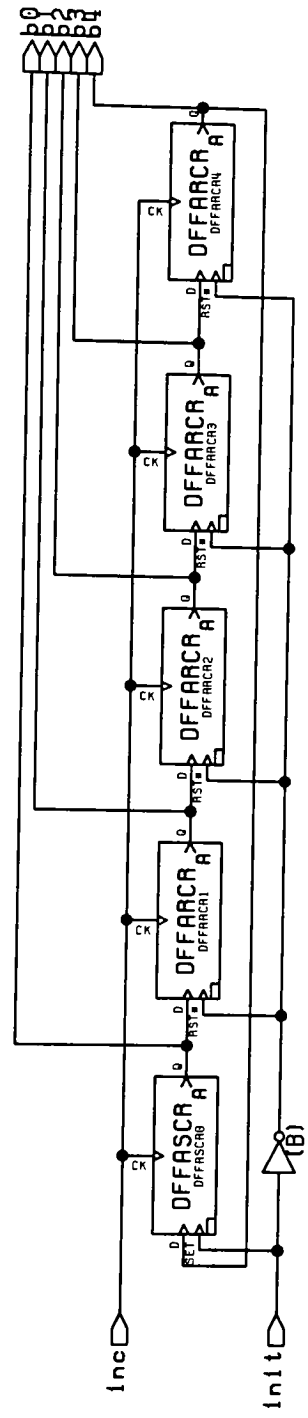

Figure 43 Logic for Prefetch Queue Counter

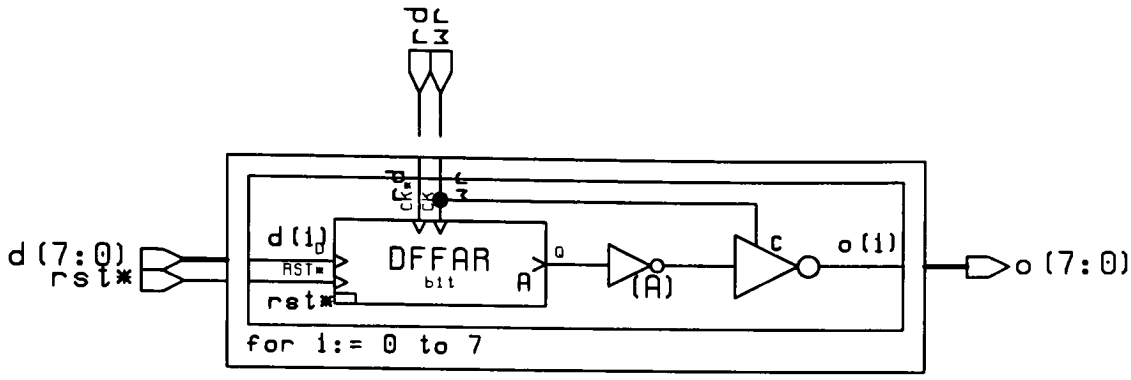

Figure 44 Logic for 8-bit Tri-State Register

The last circuit in the BU unit is the 8-bit Tri-State Register. This register is used to hold the data bytes that have been prefetched from memory. The circuit is merely a DFFAR flip-flop with a tristate inverter at its output. When a system reset occurs the *rst** line allows for the contents of the register to be cleared. The *rd* and *wr* signals are fairly intuitive. The read signal latches the data present on the *d(7:0)* bus into the first stage of the flip-flop. The *wr* signal causes the second stage to accept the data from the first stage as well as activating the tri-state inverter which presents this data onto the output bus.

3.0 Microprocessor Layout

The final step of the implementation involved creating physical mask layouts for all of the circuits used in the processor. The task of creating these layouts was made easier because of the functional block partitioning that was done earlier.

The smaller functional blocks such as the registers, the ALU and the shifter were implemented as a whole without the use of layout hierarchy. The stratgety behind the layout procedure was to obviously make the blocks as small as possible. In addition, the functional blocks were made as close to square as possible to reduce the block perimeter.

The larger functional blocks (the Control Unit and the Bus Unit) were further partitioned to make layout more managable. This allowed only the desired levels of the hierarchy to be shown, thus improving the performance of the workstation used by reducing the number of devices that needed to be constantly redrawn. As the smaller blocks were completed, they were simply connected together in a fashion that would allow for minimum area use as well as short signal line lengths.

Much time was spent on the processor floorplan. The original goal of the layout was to arrange the blocks such that minimum distances existed between the registers, ALU and mutiplier, thus reducing the time delay in the data path. However, due to the fact that the layout turned out to be very large, it was decided that simply fitting the circuitry within the pad ring would be the goal.

By placing the layout blocks in the pad ring and arranging them in a number of positions, a floorplan was finally arrived at that was a compromise between reduced line lengths and area usage. As one can see from Figure 45, there is not a great deal of space between the functional blocks. Because of this fact, signal and bus routing was

very difficult. Fortunately, since most of the blocks in the data path are lined up vertically, it was possible to run the many signals over these blocks in the second metal layer.

The processor was placed into a 4600x6800 μm^2 pad ring.

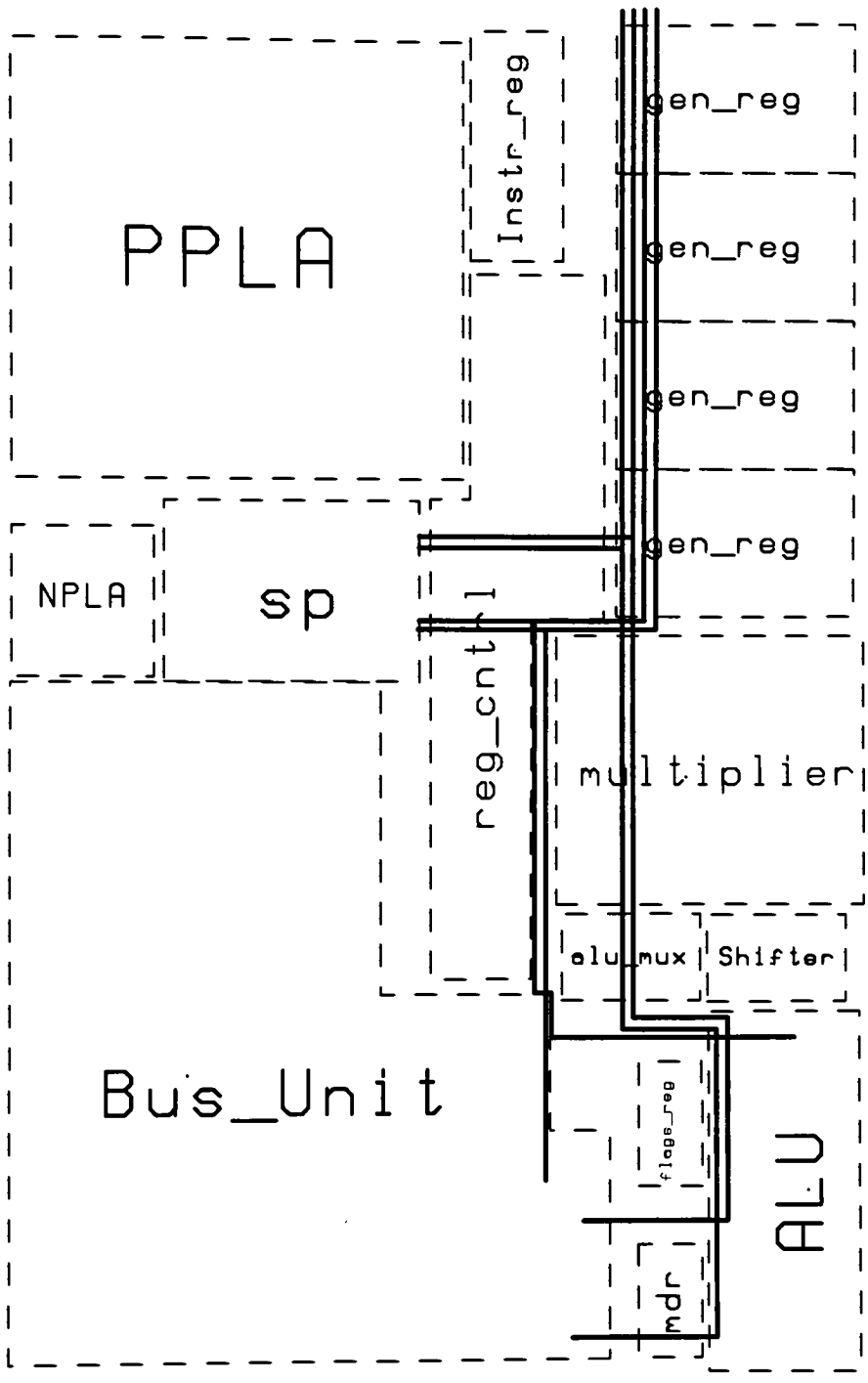

Figure 45 Microprocessor Layout Floorplan

4.0 Conclusions

Although no specific investigative goals were stated at the beginning of this paper, several important things were discovered during the course of this project.

During the instruction encoding portion of the design, it was anticipated that by using a so-called horizontal encoding of the instructions, instruction space would be wasted, but the instruction decoding would be greatly simplified. After examining the layout of the processor, it is felt that this was the correct choice since chip area is at a premium.

Probably the most important lessons learned (if not the most confounding ones) were those dealing with behavior modeling. Because of the circumstances, it was necessary to choose Mentor Graphic's BLM to model the processor. This technique uses a C language program to generate output responses for the input stimuli applied to a circuit model which is not unlike the way in which VHDL would work. However, because it is the common C language, timing of signals within the model is not automatically regulated. For example, it was necessary to be exceeding careful during modeling that when an input signal occurred, adequate time elapsed before output responses or changes in state were allowed to proceed. This problem proved to be the most troublesome in the modeling of the state machines. To use an aphromism, C used in the modeling of state machines is "magic" and must be used with the utmost care in order to ensure that the functional block does not exhibit characteristics that are physically impossible.

Upon examination of the finished layout, it was noticed that the PLAs used in the Control Unit and the Bus Unit occupy approximately 23% of the entire chip layout. Although this area is many times smaller than would be required if all of the logic were implemented with combinational logic, the PLAs are still inefficient due to their

sparse population. A more area efficient technique of implementing the control store would have been to use a microcode engine.

Although much was learned by the author from the execution of this project, it was decidedly too ambitious a project for one person to attempt in the time frame allotted.

In summary, the microprocessor incorporated approximately 12,500 transistors into the design. The nominal estimated speed was calculated to be 8 MHz without measurement of signal line capacitance which would contribute to speed reduction of the system.

5.0 References

Computer Design, Langdon, Glen G., Jr., Computeach Press Inc., 1982

Computer Architecture: A Quantitative Approach, Hennessy, John L. and Patterson, David A., Morgan Kaufmann Publishers, Inc., 1990

Rochester Institute of Technology's CMOS Standard Cell User's Manual, Lawrence Rubin, 1991

Rochester Institute of Technology's CMOS Standard Cell Administrator's Manual, Lawrence Rubin, 1991

QUICKSIM Family Reference Manual, Mentor Graphics Corporation 1989

QUICKSIM User's Manual, Mentor Graphics Corporation 1989

Accusim User's Manual, Mentor Graphics 1989

Analog Simulators Reference Manual, Mentor Graphics Corporation 1989

Appendix A:

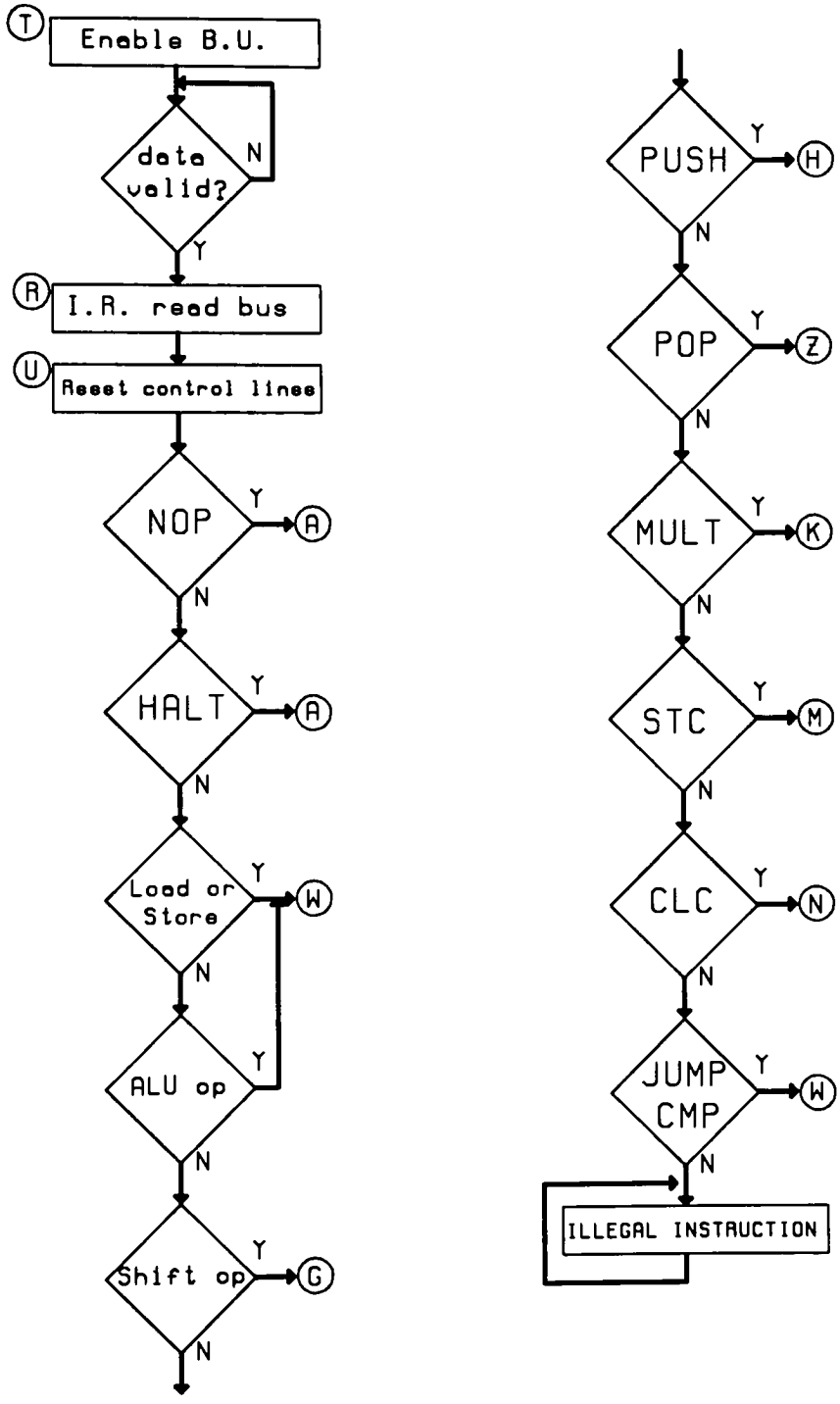

Figure A-1 Control Flow Chart of Control Unit - Pt 1

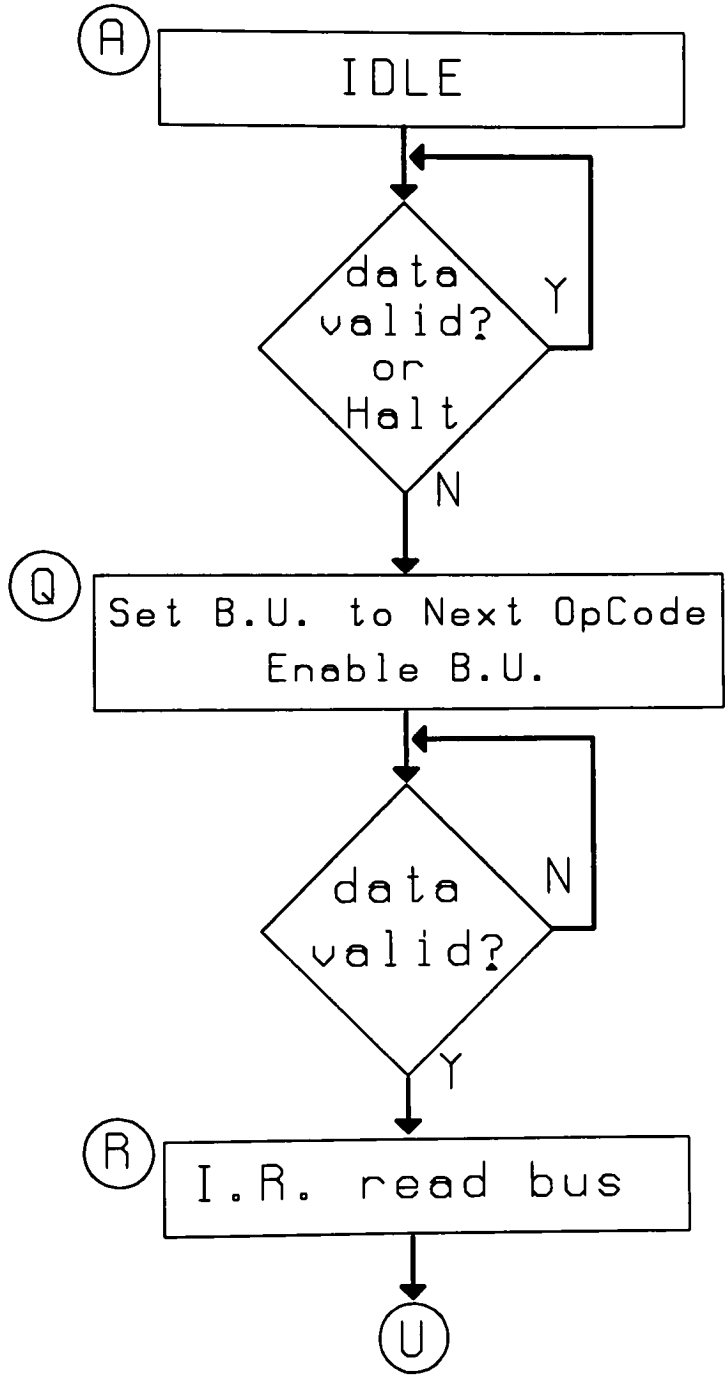

Figure A-2 Control Flow Chart of Control Unit - Pt 2

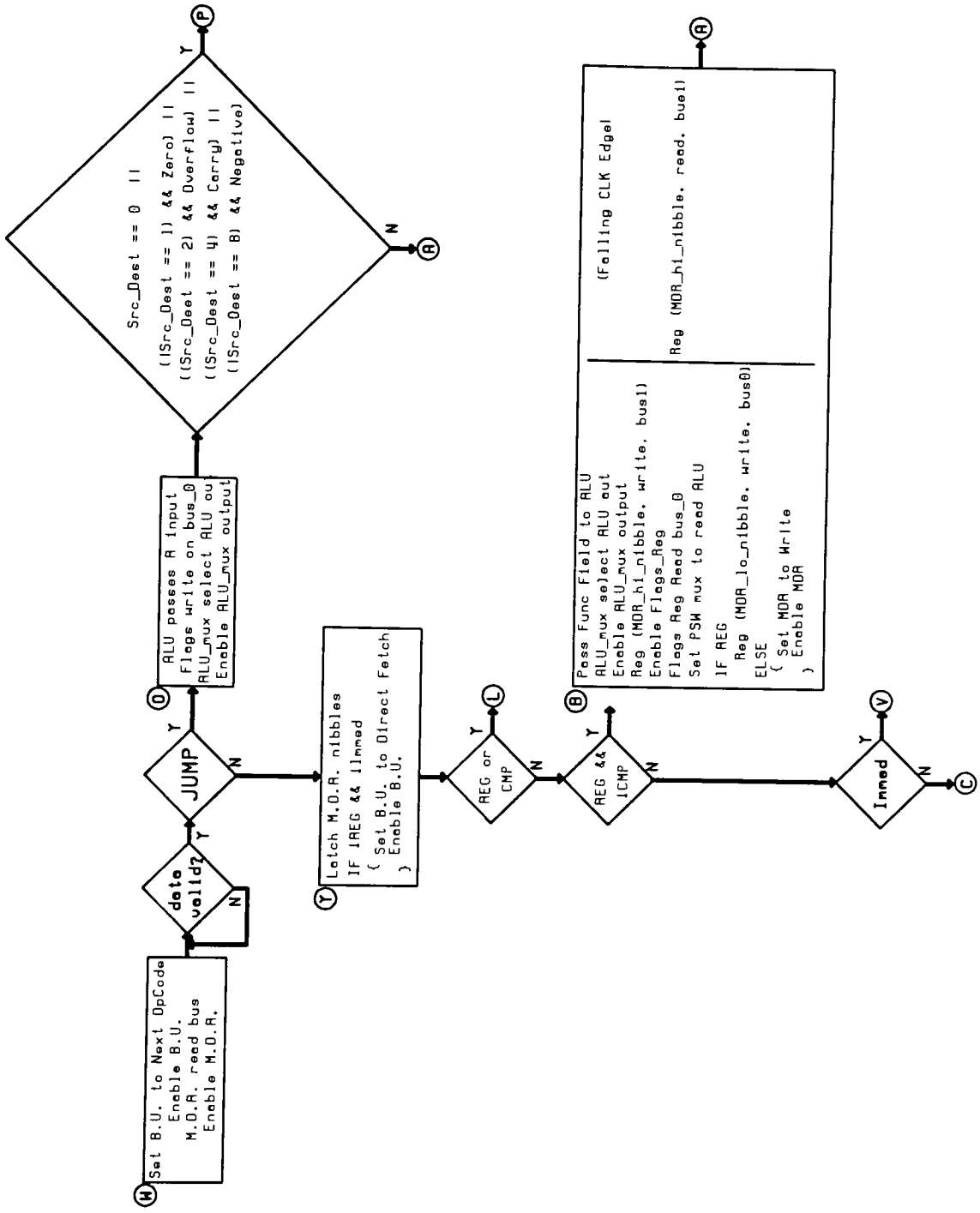

Figure A-3 Control Flow Chart of Control Unit Pt 3

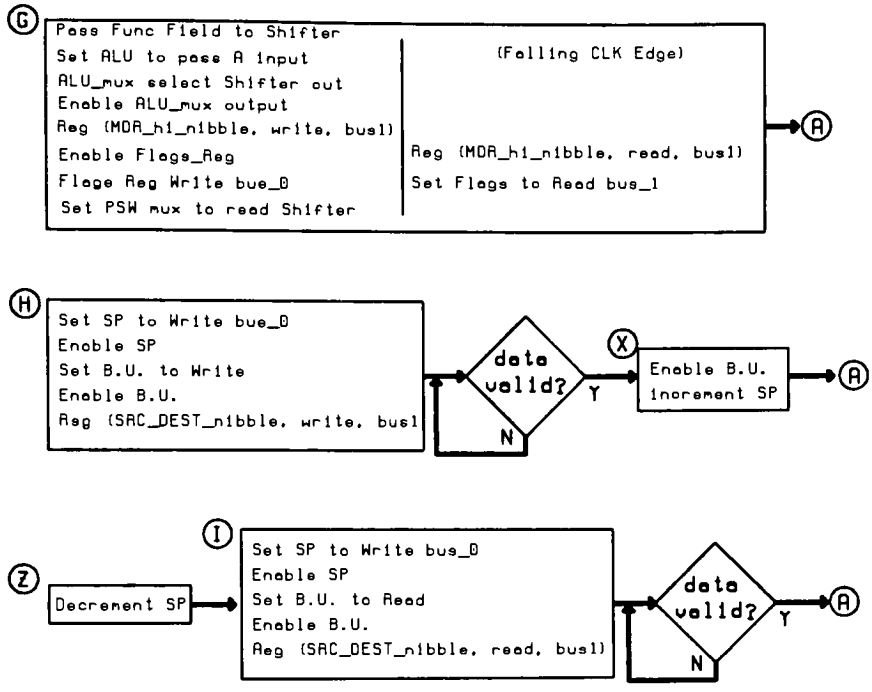

Figure A-4 Control Flow Chart of Control Unit - Pt 4

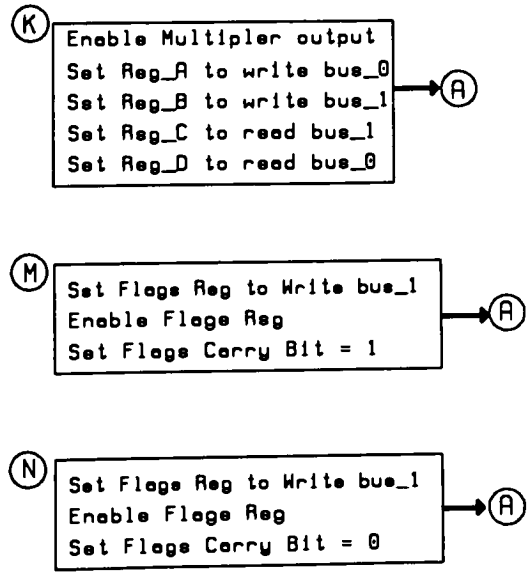

Figure A-5 Control Flow Chart of Control Unit - Pt 5

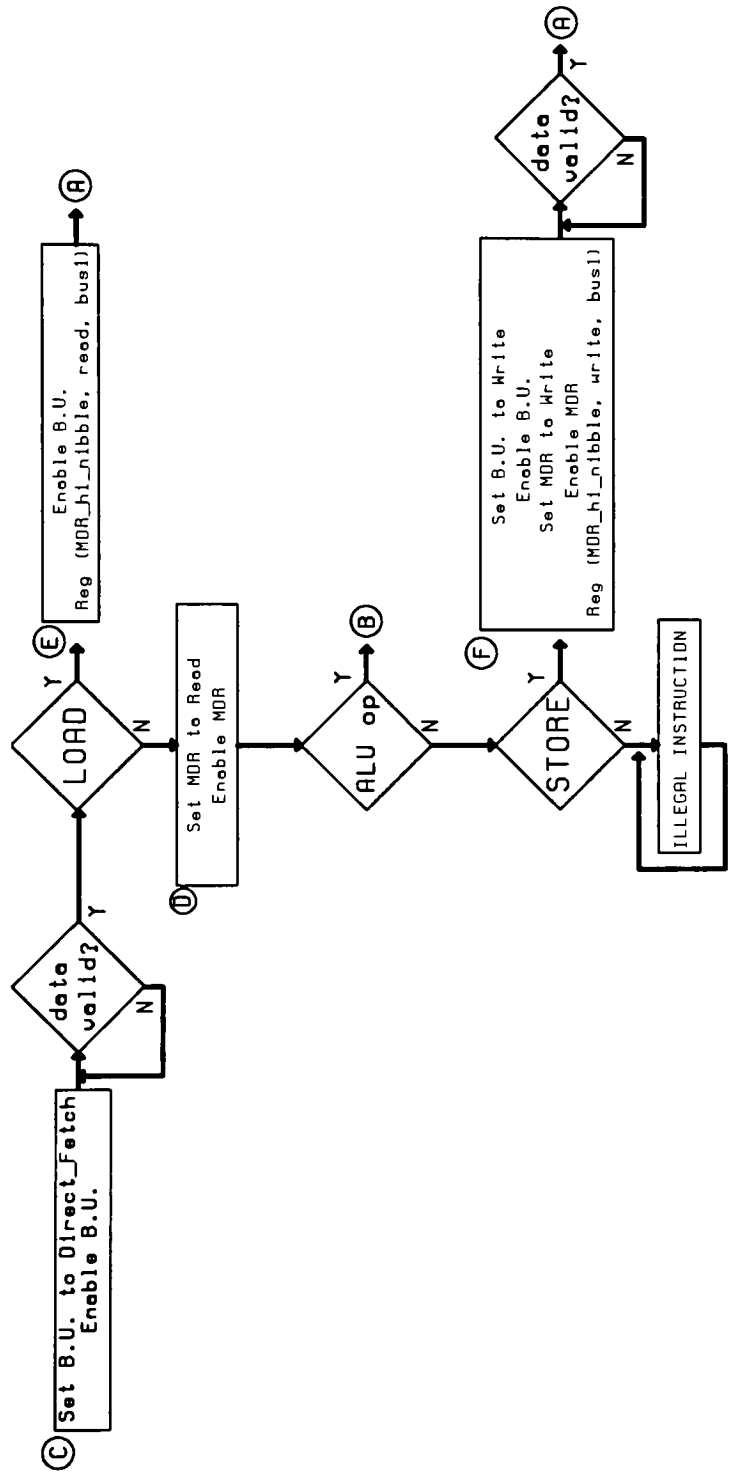

Figure A-6 Control Flow Chart of Control Unit Pt 6

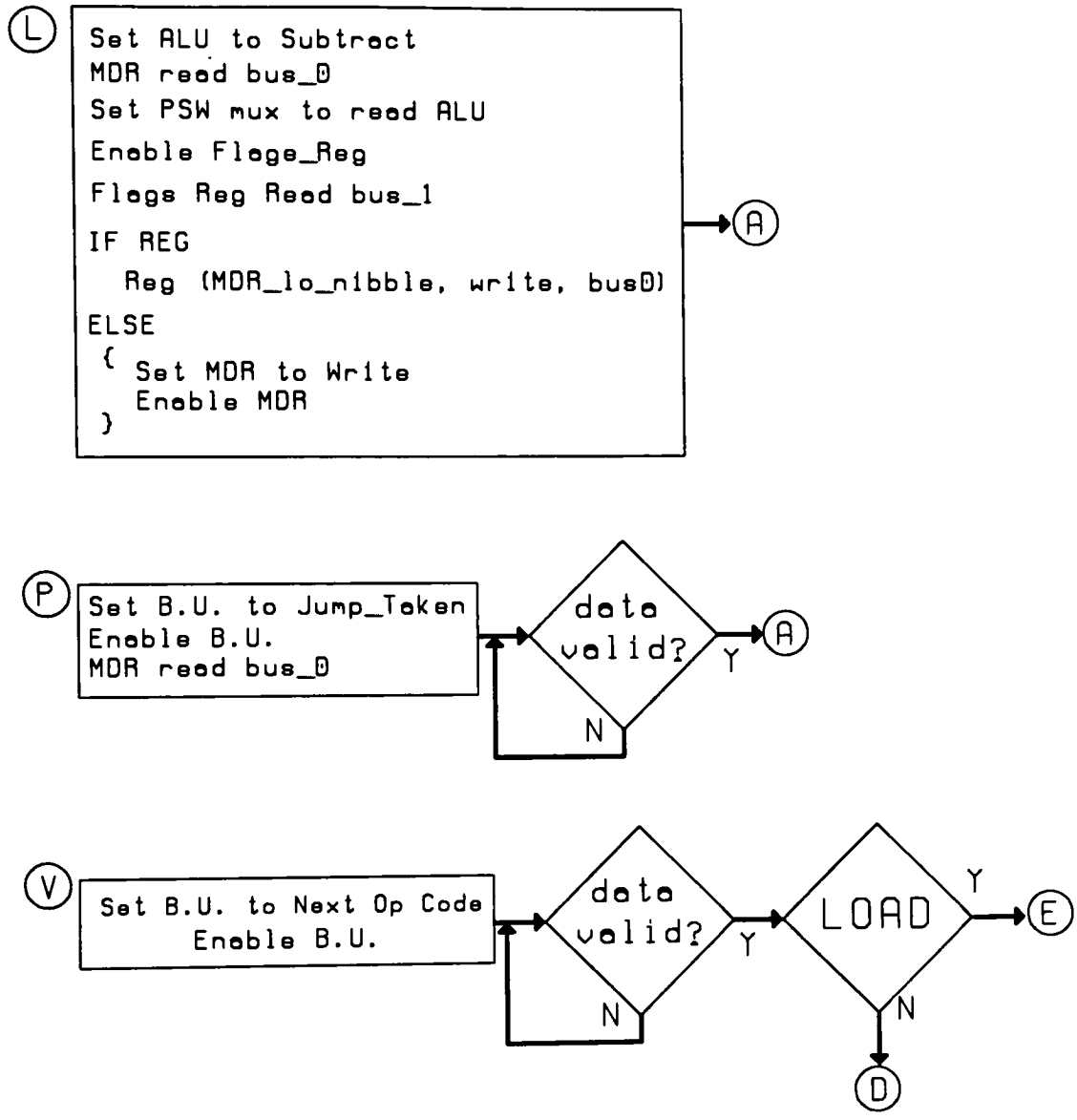

Figure A-7 Control Flow Chart of Control Unit - Pt 7

Appendix B:

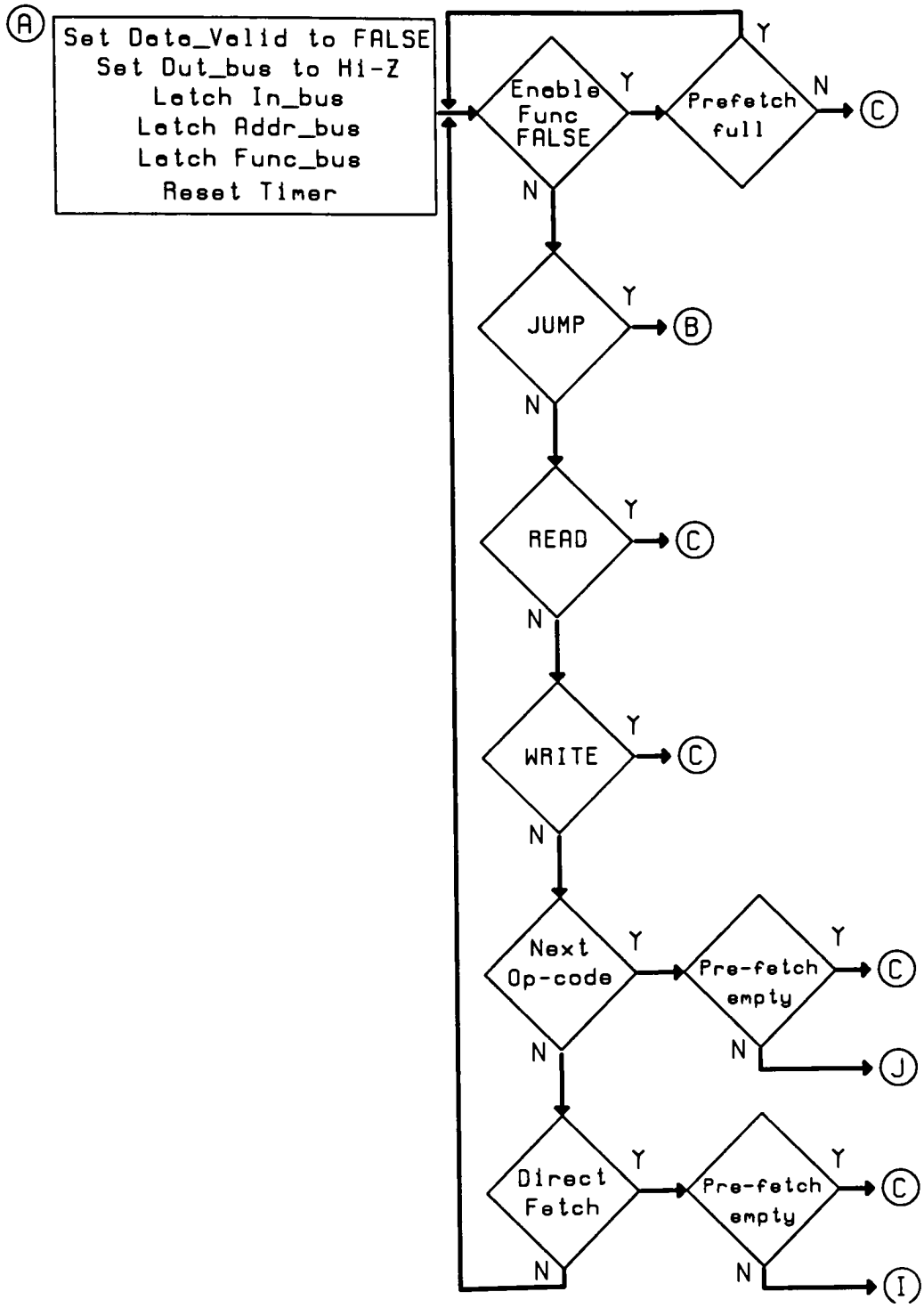

Figure B-1 Control Flow Chart of Bus Unit - Pt 1

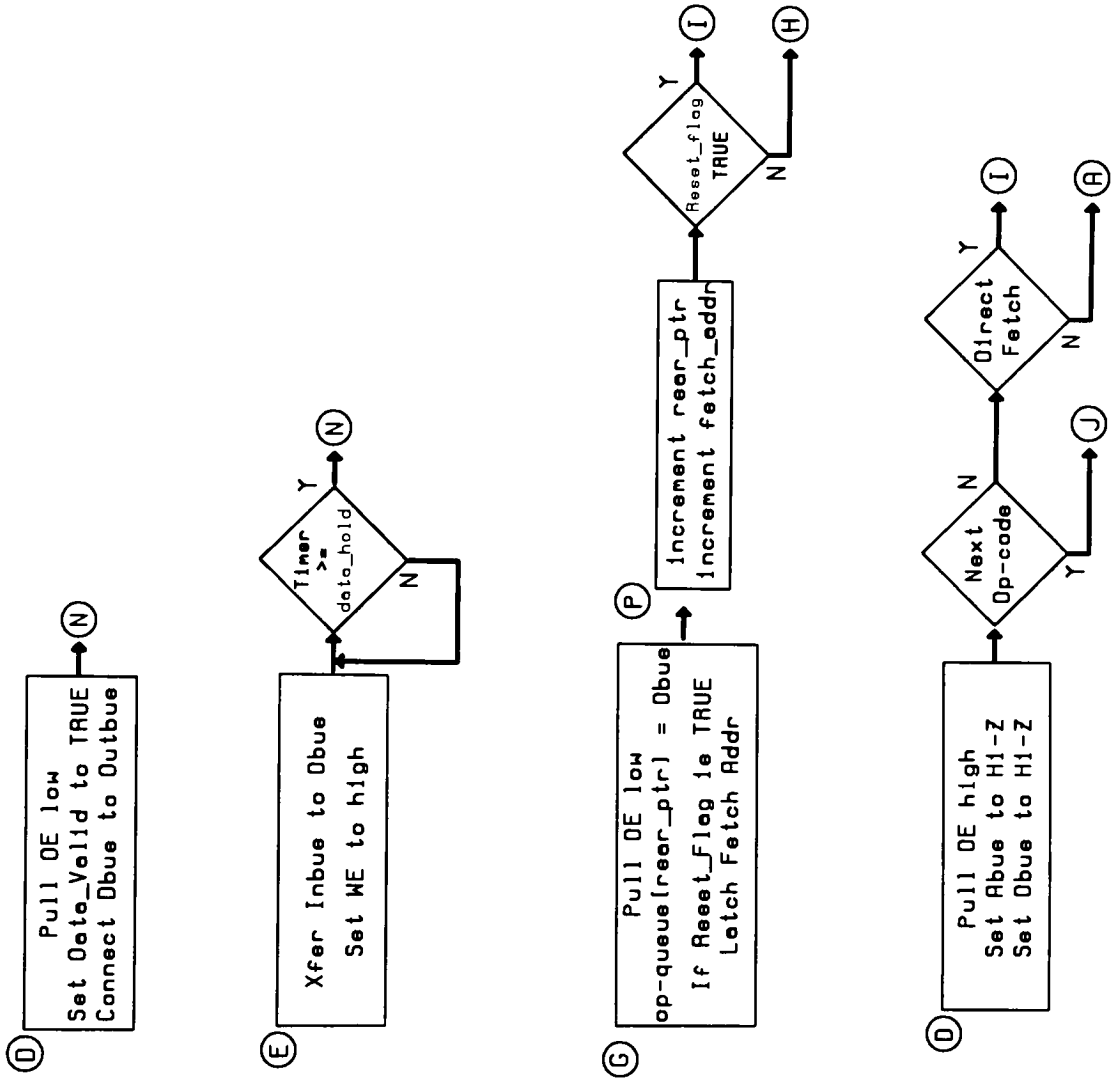

Figure B-2 Control Flow Chart of Bus Unit - Pt 2

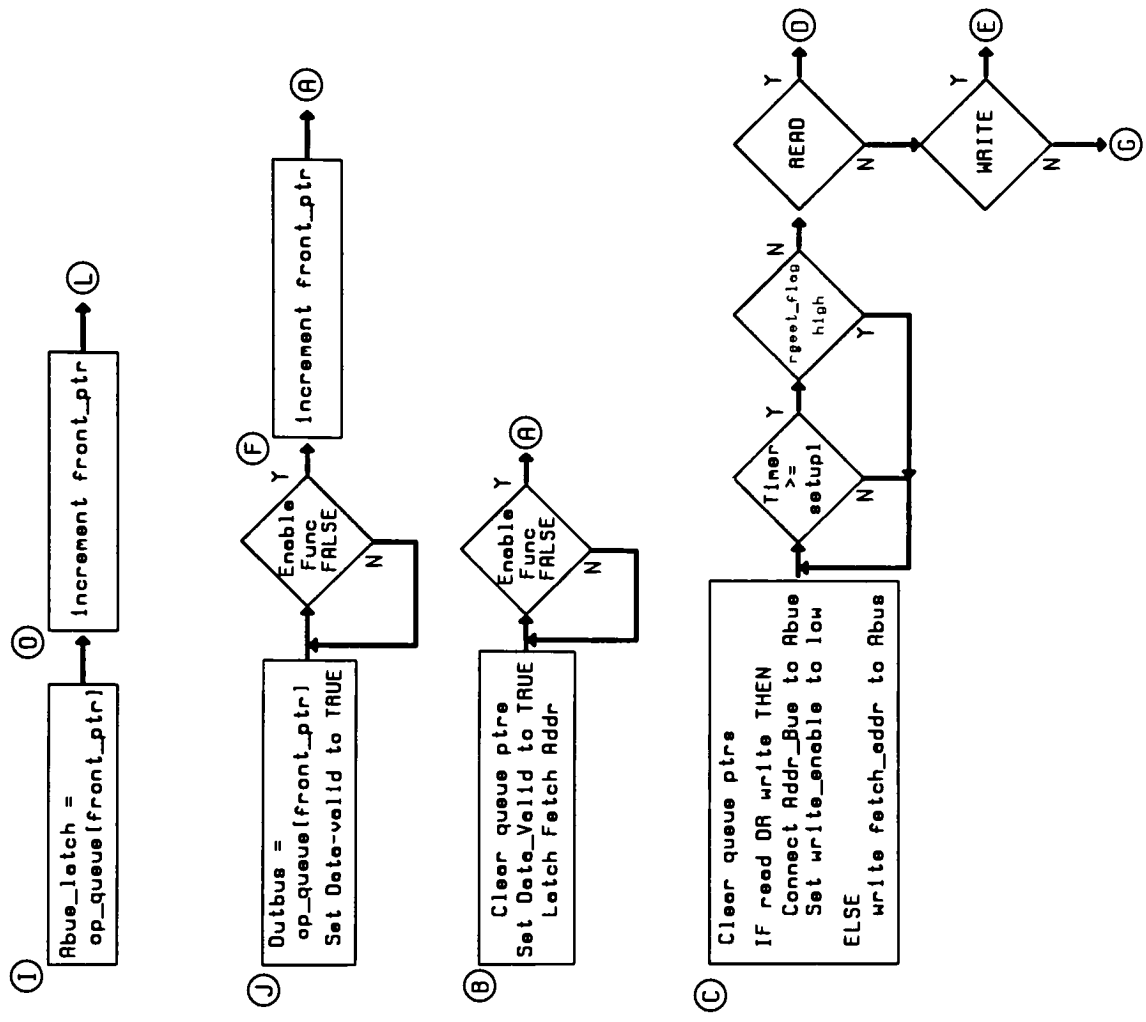

Figure B-3 Control Flow Chart of Bus Unit - Pt 3

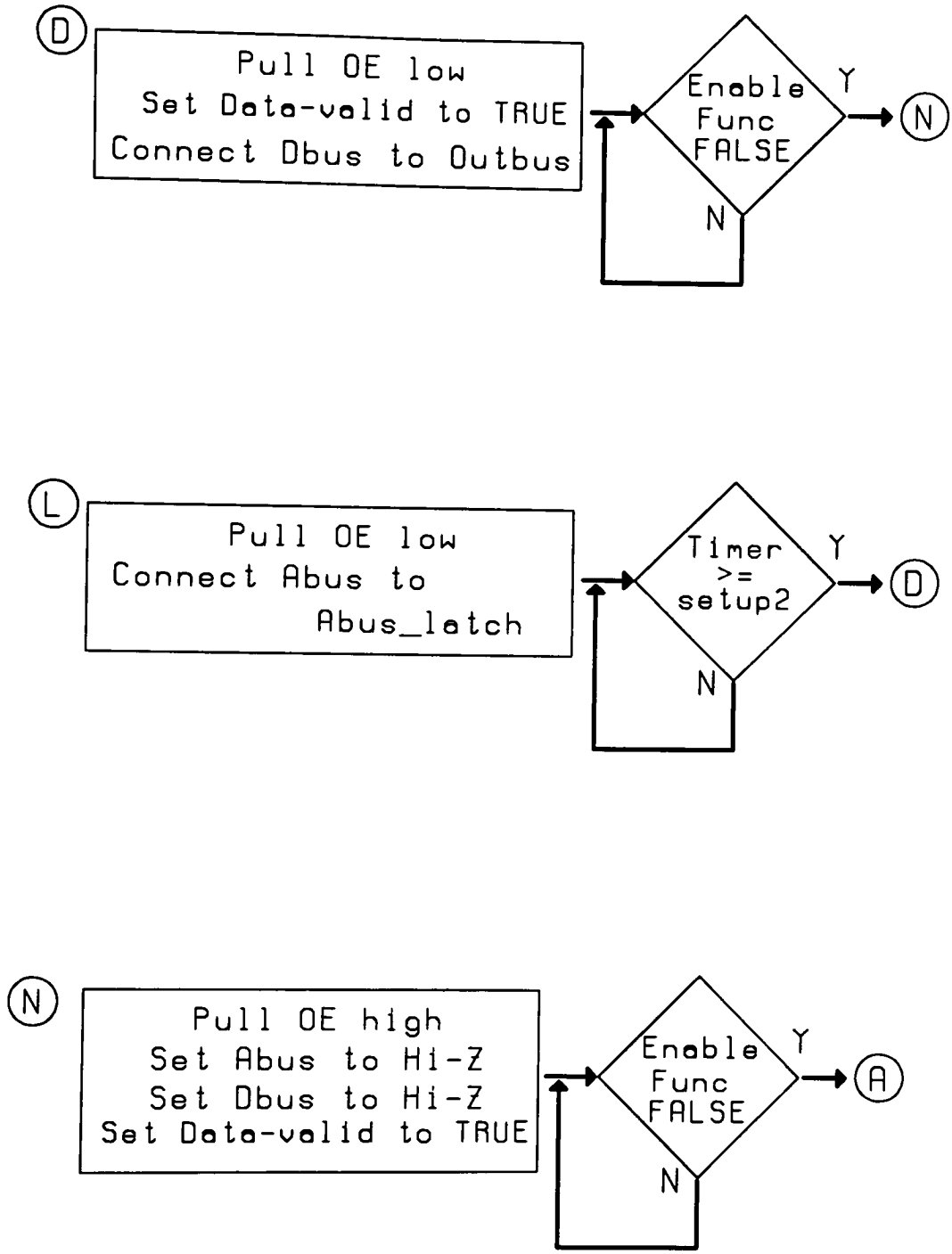

Figure B-4 Control Flow Chart of Bus Unit - Pt 4

Appendix C:

This BLM was the one used to behaviorally simulate the ALU and serves as an example of how the other models were written. Due to the large size of the BLMs for the remaining blocks within the processor, they will not be included here.

C Code for ALU BLM:

```
/*****************************************************************************
*
* Component: ALU
* Author: Jeff Correll
* Date  : 22 July 1991
*
* Comments:
*       This blm models the alu of my thesis processor
*
*****************************************************************************/

#systype "sys5"
#include <stdio.h>
#include <math.h>
#include "/idea/sys/ins/dfi.h"


/*****************************************************************************/
/*                    CONSTANTS                                              */
/*****************************************************************************/

#define  BUS_WIDTH 8     /* number of bits in the a and b busses          */
#define  PSW_WIDTH 4     /* number of bits in the psw                     */
#define  FUNC_WIDTH 4    /* number of bits in the function bus            */
#define  TRUE 1
#define  FALSE 0

                 /* These are the codes that would appear on the
                    'func' lines of the alu for the corresponding
                    operation to take place                               */
#define  ADD    0x00
#define  SUB    0x01
#define  NOT_A  0x02
#define  AND    0x03
#define  OR     0x04
```

```
#define XOR    0x05
#define NEG_A  0x06
#define CMP    0x07
#define PASS_A 0x08
#define PASS_B 0x09

/*****************************************************************************/
/*   delay times for outputs in .1nS increments                              */
/*****************************************************************************/

#define NOW   0                    /* Delay time of Zero, happens now */
#define PROP_DELAY 5                        /* token delay through alu */


/*****************************************************************************/
/*                MACROS                                                     */
/*****************************************************************************/

#define PIN_CON_VAL(pin_name,bitnum) \
(qsim_con_value[(*(qsim_instance_ptr->alu_I_\
pin_name))->bits[bitnum]])

#define PIN_OUT(pin,value,time) \
(qsim_output(&qsim_instance_ptr->alu_O_\
pin,value,time))

#define PIN_STATE(pin_name, bit_number) \
((*(qsim_instance_ptr->alu_I_\
pin_name))->bits[(bit_number)])

/*****************************************************************************/
/*                GLOBALS                                                    */
/*****************************************************************************/

static     char message[256];          /* buffer for transcript msgs    */
static     short msg_len;             /* length of msg buffer          */

char       bus_hi_z[BUS_WIDTH],   /* array filled with hi-Z values        */
           psw_hi_z[PSW_WIDTH];
char       bus_x[BUS_WIDTH],      /* array field with unknown ('X') value */
           psw_x[PSW_WIDTH];
/*****************************************************************************/

#include "/idea/sys/ins/qsim.h"
#include "/user/jac8396/thesis/blm/blm_lib.h"

#include "alu_pin.h"            /* include the pin file generated by PFGEN */
```

```
/*****************************************************************************/
alu__init()
/*****************************************************************************
*  Description:  This function is used to initialize the alu. It sets all of
*                the flags and internal variables to appropriate values.
*  Input:
*  Output:
*
*****************************************************************************/
{
   long   delay;            /* delay until output pin changes state      */
   char   level[BUS_WIDTH];  /* new level of changing output pins        */
   int    cnt;              /* counter variable                          */
   long   time;             /* holds present simulation time             */

   dfi_$object_identifier     inst_id;
   dfi_$inst_rec_ptr          inst_ptr;

   inst_id = qsim_get_instance_id();
   dfi_$get_inst(&qsim_dfi_channel, &inst_id, &inst_ptr);

   strncpy(ID_String, inst_ptr->instance_pathname.chars, inst_ptr->instance_pathname.len);

   for (cnt=0; cnt<BUS_WIDTH; cnt++)  /* set hi-z and unknown (x) array to   */
   {                          /*  appropriate values                       */
      bus_hi_z[cnt] = QSIM_XZ;
      bus_x[cnt] = QSIM_XS;
   }

   for (cnt=0; cnt<PSW_WIDTH; cnt++)
   {
      psw_hi_z[cnt] = QSIM_XZ;
      psw_x[cnt] = QSIM_XS;
   }

   delay = NOW;               /* init the outputs to unknown, now          */
   PIN_OUT(out, bus_x, &delay);
   PIN_OUT(psw, bus_x, &delay);
}
/*****************************************************************************/

/*****************************************************************************/
alu__a()
/*****************************************************************************
*  Description:  Pin handler for input bus a
*  Input: Data from the input bus a
*  Output:
```

```
*****************************************************************************/
{
   do_alu_func();
}
/*****************************************************************************/
/*****************************************************************************/
alu__b()
/*****************************************************************************
*  Description:  Pin handler for input bus b
*  Input: Data from the input bus a
*  Output:
*****************************************************************************/
{
   do_alu_func();
}
/*****************************************************************************/
/*****************************************************************************/
alu__func()
/*****************************************************************************
*  Description:  Pin handler for function input bus
*  Input: Data from the input bus func
*  Output:
*****************************************************************************/
{
   do_alu_func();
}
/*****************************************************************************/

/*****************************************************************************/
do_alu_func()
/*****************************************************************************
*  Description:  This function performs all of the functions of the alu. These
*                include normal alu functions such as ADD, SUB, etc as well
*                as setting up the PSW word.
*  Input:
*  Output:
*****************************************************************************/
{

   char   abus[BUS_WIDTH],   /* these arrays hold the Quicksim values of   */
          bbus[BUS_WIDTH],   /*  the input busses                          */
          out[BUS_WIDTH];
   char   psw[PSW_WIDTH],
          func[FUNC_WIDTH];

   int    bit;            /* used as a counter for each bit in the array  */
   int    a_error,        /* error in conversion flags                    */
```

```
        b_error,
        func_error;

    int   abus_int=0,      /* integer value of input and output buses     */
        bbus_int=0,
        func_int=0,
        psw_int=0,
        out_int=0;

    int   zero=0,        /* integer values of psw bits                  */
        carry=0,
        negative=0,
        overflow=0;


    long   delay=PROP_DELAY;  /* delay in output signal appearance        */

    for(bit=0; bit < BUS_WIDTH; bit++)  /* read a-bus and b-bus qsim values  */
    {
       abus[bit] = PIN_STATE(a,bit);
       bbus[bit] = PIN_STATE(b,bit);
    }

    for(bit=0; bit < FUNC_WIDTH; bit++) /* read the func bus qsim values     */
       func[bit] = PIN_STATE(func,bit);


    a_error = qstoi(abus, BUS_WIDTH, &abus_int); /* convert qsim value to int */
    if (a_error)
       qmsg("abus not convertible...");

    b_error = qstoi(bbus, BUS_WIDTH, &bbus_int); /* convert qsim value to int */
    if (b_error)
       qmsg("bbus not convertible...");

    func_error = qstoi(func,FUNC_WIDTH, &func_int);/* convert qsim value to int */
    if (func_error)
       qmsg("func not convertible...");

qmsg("func_int = 0x%2.2x",func_int);
   if (!func_error)
       switch(func_int)        /* perform correct alu function              */
       {
          case ADD:
             qmsg("ADD");
             out_int = abus_int + bbus_int;
             break;
```

```
case SUB:
   qmsg("SUB");
   out_int = abus_int - bbus_int;
   break;

case AND:
   qmsg("AND");
   out_int = abus_int & bbus_int;
   break;

case OR:
   qmsg("OR");
   out_int = abus_int | bbus_int;
   break;

case NOT_A:
   qmsg("NOT_A");
   out_int = ~abus_int;
   break;

case XOR:
   qmsg("XOR");
   out_int = abus_int ^ bbus_int;
   break;

case NEG_A:
   qmsg("NEG_A");
   out_int = (abus_int*(-1))&0xFF;
   break;

case CMP:
   qmsg("CMP");
   break;

case PASS_A:
   qmsg("PASS_A");
   out_int = abus_int;
   qmsg("Passing 0x%2.2x",abus_int);
   break;

case PASS_B:
   qmsg("PASS_B");
   out_int = bbus_int;
   qmsg("Passing 0x%2.2x",bbus_int);
   break;
```

```
        default:
           qmsg("Illegal function");
           break;
      }

qmsg("out_int = 0x%2.2x",out_int);

      if ( ((out_int > 0xFF) || (out_int < 0x00))   /* should carry be set    */
          && ( (func_int == NEG_A)||(func_int == ADD)||
              (func_int == SUB)||(func_int == CMP)
             )
         )
          carry = 1;

      if (out_int & 0x80)        /* should negative be set                */
          negative = 1;

      if (((abus_int & 0x80) == (bbus_int & 0x80)) && carry)
          overflow = 1;

      if ((out_int & 0xff) == 0x00)   /* should zero be set               */
          zero = 1;

      out_int = out_int & 0xFF;  /* keep only the 8 least significant bits   */

   /* if there are no conversion errors or there are errors converting
        the b-bus when the function is only a-bus related output the
        results else put out unknowns on the output bus                    */

      if ((b_error && (func_int == PASS_A || func_int == NOT_A ||
                       func_int == NEG_A
                      )
          ) || !(a_error || b_error || func_error)
      ){
          itoqs(out_int,BUS_WIDTH,out);
          PIN_OUT(out,out,&delay);

          psw_int = (negative*8) + (zero*4) +  /* set up the psw (NZVC)      */
                    (overflow*2) + carry;
          itoqs(psw_int,PSW_WIDTH,psw);
          PIN_OUT(psw,psw,&delay);
qmsg("PSW=0x%2.2x",psw_int);
      }
      else
      {
          PIN_OUT(out,bus_x,&delay);
          PIN_OUT(psw,psw_x,&delay);
```

```
        }
    }
/*****************************************************************************/
```

Appendix D:

Instruction Register Logical Simulation Results:

```
  0.0 XXr  0  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X
200.0 00   0  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X
250.0 00   1  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X
300.0 00   0  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X
309.0 00   0  X  X  X  X  X  X  X  X  X  X  X  X  X  X  0  0
311.0 00   0  0  X  0  X  0  0  0  0  0  X  0  0  0  0  X  0  0
312.0 00   0  0  1  0  1  0  0  0  0  0  X  0  0  0  0  X  0  0
313.0 00   0  0  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0
400.0 40   0  0  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  ADD (immed)
450.0 40   1  0  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0
500.0 40   0  0  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0
511.0 40   0  0  1  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0
512.0 40   0  0  1  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0
600.0 42   0  0  1  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  ADD (direct)
650.0 42   1  0  1  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0
700.0 42   0  0  1  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0
709.0 42   0  0  1  0  0  1  0  0  0  0  0  0  0  0  0  0  0  2
710.0 42   0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  2
800.0 44   0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  2  ADD (reg)
850.0 44   1  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  2
900.0 44   0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  2
909.0 44   0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  4
910.0 44   0  0  1  0  0  1  0  0  0  0  0  0  0  0  0  0  0  4
911.0 44   0  1  1  0  0  1  0  0  0  0  0  0  0  0  0  0  0  4
912.0 44   0  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  4
1000.0 48  0  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  4  SUB (immed)
1050.0 48  1  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  4
1100.0 48  0  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  4
1109.0 48  0  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  1  0
1111.0 48  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  1  0
1112.0 48  0  0  1  0  0  1  0  0  0  0  0  0  0  0  0  0  1  0
1200.0 50  0  0  1  0  0  1  0  0  0  0  0  0  0  0  0  0  1  0  NOT (immed)
1250.0 50  1  0  1  0  0  1  0  0  0  0  0  0  0  0  0  0  1  0
1300.0 50  0  0  1  0  0  1  0  0  0  0  0  0  0  0  0  0  1  0
1309.0 50  0  0  1  0  0  1  0  0  0  0  0  0  0  0  0  0  2  0

 TIME  ^ibus      ^immed   ^alu_op  ^shift_op      ^clc     ^instr
           ^read     ^halt     ^load     ^push ^mult     ^jump     ^src_dest
              ^reg      ^nop      ^store    ^pop  ^stc      ^illegal
```

1400.0	58	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	2	0	AND (immed)
1450.0	58	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	2	0	
1500.0	58	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	2	0	
1509.0	58	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	3	0	
1600.0	60	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	3	0	OR (immed)
1650.0	60	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	3	0	
1700.0	60	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	3	0	
1709.0	60	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	4	0	
1800.0	68	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	4	0	XOR (immed)
1850.0	68	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	4	0	
1900.0	68	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	4	0	
1909.0	68	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	5	0	
2000.0	70	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	5	0	NEG (immed)
2050.0	70	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	5	0	
2100.0	70	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	5	0	
2109.0	70	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	6	0	
2200.0	78	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	6	0	CMP (immed)
2250.0	78	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	6	0	
2300.0	78	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	6	0	
2309.0	78	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	7	0	
2400.0	08	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	7	0	LD (immed)
2450.0	08	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	7	0	
2500.0	08	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	7	0	
2509.0	08	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	
2511.0	08	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	
2512.0	08	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	
2600.0	10	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	ST (immed)
2650.0	10	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	
2700.0	10	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	
2709.0	10	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	2	0	
2710.0	10	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	
2711.0	10	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	
2800.0	C0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	LSR (shift)
2850.0	C0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	
2900.0	C0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	
2909.0	C0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
2911.0	C0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
3000.0	C8	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	LSL (shift)
3050.0	C8	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
3100.0	C8	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
3109.0	C8	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	

```
TIME  ^ibus     ^immed   ^alu_op  ^shift_op      ^clc     ^instr
          ^read    ^halt    ^load    ^push ^mult    ^jump    ^src_dest
             ^reg     ^nop     ^store   ^pop  ^stc     ^illegal
```

```
3200.0 D0  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  1  0  ASR (shift)
3250.0 D0  1  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  1  0
3300.0 D0  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  1  0
3309.0 D0  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  2  0
3400.0 D8  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  2  0  ASL (shift)
3450.0 D8  1  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  2  0
3500.0 D8  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  2  0
3509.0 D8  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  3  0
3600.0 E0  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  3  0  ROR (shift)
3650.0 E0  1  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  3  0
3700.0 E0  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  3  0
3709.0 E0  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  4  0
3800.0 E8  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  4  0  ROL (shift)
3850.0 E8  1  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  4  0
3900.0 E8  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  4  0
3909.0 E8  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  5  0
4000.0 88  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  5  0  Push
4050.0 88  1  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  5  0
4100.0 88  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  5  0
4109.0 88  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  1  0
4111.0 88  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0
4112.0 88  0  0  1  0  0  0  0  0  0  1  0  0  0  0  0  0  1  0
4200.0 80  0  0  1  0  0  0  0  0  0  1  0  0  0  0  0  0  1  0  Pop
4250.0 80  1  0  1  0  0  0  0  0  0  1  0  0  0  0  0  0  1  0
4300.0 80  0  0  1  0  0  0  0  0  0  1  0  0  0  0  0  0  1  0
4309.0 80  0  0  1  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0
4310.0 80  0  0  1  0  0  0  0  0  0  1  1  0  0  0  0  0  0  0
4311.0 80  0  0  1  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0
4400.0 98  0  0  1  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  MULT
4450.0 98  1  0  1  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0
4500.0 98  0  0  1  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0
4509.0 98  0  0  1  0  0  0  0  0  0  0  1  0  0  0  0  0  3  0
4510.0 98  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  3  0
4511.0 98  0  0  1  0  0  0  0  0  0  0  0  1  0  0  0  0  3  0
4600.0 B0  0  0  1  0  0  0  0  0  0  0  0  1  0  0  0  0  3  0  STC
4650.0 B0  1  0  1  0  0  0  0  0  0  0  0  1  0  0  0  0  3  0
4700.0 B0  0  0  1  0  0  0  0  0  0  0  0  1  0  0  0  0  3  0
4709.0 B0  0  0  1  0  0  0  0  0  0  0  0  1  0  0  0  0  6  0
4710.0 B0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  6  0
4711.0 B0  0  0  1  0  0  0  0  0  0  0  0  0  1  0  0  0  6  0

 TIME  ^ibus      ^immed   ^alu_op  ^shift_op      ^clc      ^instr
           ^read     ^halt     ^load     ^push ^mult     ^jump     ^src_dest
              ^reg      ^nop      ^store    ^pop  ^stc      ^illegal
```

```
4800.0 A8  0  0  1  0  0  0  0  0  0  0  0  0  1  0  0  0  6  0  CLC
4850.0 A8  1  0  1  0  0  0  0  0  0  0  0  0  1  0  0  0  6  0
4900.0 A8  0  0  1  0  0  0  0  0  0  0  0  0  1  0  0  0  6  0
4909.0 A8  0  0  1  0  0  0  0  0  0  0  0  0  1  0  0  0  5  0
4910.0 A8  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  5  0
4911.0 A8  0  0  1  0  0  0  0  0  0  0  0  0  0  1  0  0  5  0
5000.0 B8  0  0  1  0  0  0  0  0  0  0  0  0  0  1  0  0  5  0  Jump Always
5050.0 B8  1  0  1  0  0  0  0  0  0  0  0  0  0  1  0  0  5  0
5100.0 B8  0  0  1  0  0  0  0  0  0  0  0  0  0  1  0  0  5  0
5109.0 B8  0  0  1  0  0  0  0  0  0  0  0  0  0  1  0  0  7  0
5110.0 B8  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  7  0
5111.0 B8  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  0
5200.0 B9  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  0  JUMP zero
5250.0 B9  1  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  0
5300.0 B9  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  0
5309.0 B9  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  1
5400.0 BA  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  1  JUMP ovflw
5450.0 BA  1  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  1
5500.0 BA  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  1
5509.0 BA  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  2
5510.0 BA  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  2
5600.0 BB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  2  JUMP carry
5650.0 BB  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  2
5700.0 BB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  2
5709.0 BB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  3
5800.0 BC  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  3  JUMP neg
5850.0 BC  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  3
5900.0 BC  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  3
5909.0 BC  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  4
5910.0 BC  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  4
6000.0 BD  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  4  JUMP a>b
6050.0 BD  1  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  4
6100.0 BD  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  4
6109.0 BD  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  5
6200.0 BE  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  5  JUMP a >= b
6250.0 BE  1  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  5
6300.0 BE  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  5
6309.0 BE  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  7  6
6310.0 BE  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  6

 TIME  ^ibus      ^immed   ^alu_op  ¨shift_op      ^clc     ^instr
          ¨read    ^halt    ^load    ^push ^mult    ^jump    ^src_dest
             ^reg     ^nop     ^store   ^pop  ^stc     ^illegal
```

```
6400.0 BF  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  6  JUMP a <> b
6450.0 BF  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  6
6500.0 BF  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  6
6509.0 BF  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  7
6600.0 A0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  7  HALT
6650.0 A0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  7
6700.0 A0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  7
6709.0 A0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  4  0
6710.0 A0  0  0  1  1  0  0  0  0  0  0  0  0  1  1  1  0  4  0
6711.0 A0  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  4  0
6800.0 FF  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  4  0  Illegal op
6850.0 FF  1  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  4  0
6900.0 FF  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  4  0
6909.0 FF  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  7  7
6910.0 FF  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  7  7
6911.0 FF  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  7  7
6912.0 FF  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  7  7
6915.0 FF  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  7  7

       TIME  ^ibus      ^immed   ^alu_op  ^shift_op      ¨clc     ^instr
                ^read     ^halt    ^load    ^push ¨mult    ^jump    ^src_dest
                   ^reg      ¨nop     ¨store   ¨pop  ^stc     ^illegal
```

ALU Logical Simulation Results:

```
  0.0 0  00  00  X  XX  ADD
  5.0 0  00  00  X  XX
 12.0 0  00  00  X  XX
 13.0 0  00  00  X  XX
 16.0 0  00  00  X  00
 19.0 0  00  00  X  00
 20.0 0  00  00  4  00
100.0 0  FF  FF  4  00
104.0 0  FF  FF  6  00
107.0 0  FF  FF  X  XX
109.0 0  FF  FF  6  00
110.0 0  FF  FF  X  00
111.0 0  FF  FF  X  XX
112.0 0  FF  FF  X  XX
113.0 0  FF  FF  X  FE
114.0 0  FF  FF  X  FE
116.0 0  FF  FF  X  FE
118.0 0  FF  FF  8  FE
150.0 0  00  80  8  FE
158.0 0  00  80  X  XX
159.0 0  00  80  X  XE
160.0 0  00  80  8  FE
161.0 0  00  80  X  X0
162.0 0  00  80  8  XX
163.0 0  00  80  8  80  <--

TIME  ^func  ^b     ^out
        ^a       ^psw
```

```
200.0 1  00  00  8  80  SUB
207.0 1  00  00  8  XX
208.0 1  00  00  8  FF
209.0 1  00  00  8  FE
213.0 1  00  00  8  FC
217.0 1  00  00  8  F8
221.0 1  00  00  8  F0
225.0 1  00  00  8  E0
229.0 1  00  00  8  C0
233.0 1  00  00  8  80
237.0 1  00  00  0  00
240.0 1  00  00  4  00
250.0 1  FF  FF  4  00
257.0 1  FF  FF  X  XX
259.0 1  FF  FF  4  00
260.0 1  FF  FF  X  00
262.0 1  FF  FF  4  00
300.0 1  01  AA  4  00
308.0 1  01  AA  X  XX
309.0 1  01  AA  X  XX
310.0 1  01  AA  C  AB
311.0 1  01  AA  C  XX
312.0 1  01  AA  X  XX
313.0 1  01  AA  8  FF
316.0 1  01  AA  X  XX
318.0 1  01  AA  0  57  <--

TIME  ^func  ^b     ^out
        ^a       ^psw
```

```
350.0 2  00  AA  0  57  NOT
356.0 2  00  AA  X  XX
357.0 2  00  AA  8  AX
358.0 2  00  AA  8  A9
359.0 2  00  AA  8  A8
360.0 2  00  AA  X  XX
362.0 2  00  AA  X  5X
363.0 2  00  AA  X  XX
364.0 2  00  AA  X  XX
365.0 2  00  AA  9  A3
366.0 2  00  AA  X  A3
367.0 2  00  AA  X  XX
368.0 2  00  AA  X  XX
369.0 2  00  AA  0  47
370.0 2  00  AA  X  47
371.0 2  00  AA  X  XX
372.0 2  00  AA  X  XX
373.0 2  00  AA  9  8F
374.0 2  00  AA  X  8F
375.0 2  00  AA  X  XF
376.0 2  00  AA  X  XF
377.0 2  00  AA  0  1F
378.0 2  00  AA  X  1F
379.0 2  00  AA  X  XF
380.0 2  00  AA  1  XF
381.0 2  00  AA  1  3F
383.0 2  00  AA  1  XF
384.0 2  00  AA  X  XF
385.0 2  00  AA  1  7F
387.0 2  00  AA  X  XF
389.0 2  00  AA  9  FF
390.0 2  00  AA  X  FF
392.0 2  00  AA  8  FF <--
400.0 2  FF  AA  8  FF
408.0 2  FF  AA  X  XX
410.0 2  FF  AA  0  00
411.0 2  FF  AA  X  00
413.0 2  FF  AA  4  00 <--

TIME  ^func  ^b    ^out
        ^a     ^psw
```

```
450.0 3  55  AA  4  00  AND
457.0 3  55  AA  X  XX
458.0 3  55  AA  4  55
459.0 3  55  AA  4  XX
461.0 3  55  AA  4  54
462.0 3  55  AA  X  AA
463.0 3  55  AA  X  XX
464.0 3  55  AA  0  00
467.0 3  55  AA  4  00 <--
500.0 3  F4  42  4  00
507.0 3  F4  42  X  X0
508.0 3  F4  42  X  X0
509.0 3  F4  42  X  40
510.0 3  F4  42  X  40
511.0 3  F4  42  X  40
512.0 3  F4  42  0  40 <--
550.0 4  AA  55  0  40  OR
557.0 4  AA  55  0  0X
558.0 4  AA  55  8  XX
559.0 4  AA  55  8  XE
560.0 4  AA  55  9  FF <--
600.0 4  FF  FF  9  FF <--
650.0 5  AA  55  9  FF  XOR
657.0 5  AA  55  1  01
658.0 5  AA  55  X  XX
659.0 5  AA  55  9  XX
660.0 5  AA  55  9  FF
661.0 5  AA  55  8  FF <--
700.0 5  AA  AA  8  FF
707.0 5  AA  AA  8  XX
708.0 5  AA  AA  X  XX
709.0 5  AA  AA  0  00
711.0 5  AA  AA  1  00
712.0 5  AA  AA  5  00 <--

TIME  ^func  ^b    ^out
        ^a     ^psw
```

```
750.0 6  00  AA  5  00  NEG
758.0 6  00  AA  X  XX
759.0 6  00  AA  X  XX
760.0 6  00  AA  X  XX
761.0 6  00  AA  X  XX
762.0 6  00  AA  X  XX
763.0 6  00  AA  X  XX
764.0 6  00  AA  X  XX
765.0 6  00  AA  X  A8
766.0 6  00  AA  X  XX
767.0 6  00  AA  X  XX
768.0 6  00  AA  X  X0
769.0 6  00  AA  X  50
770.0 6  00  AA  X  X0
771.0 6  00  AA  X  X0
772.0 6  00  AA  X  X0
773.0 6  00  AA  X  A0
774.0 6  00  AA  X  X0
775.0 6  00  AA  X  X0
776.0 6  00  AA  X  X0
777.0 6  00  AA  X  40
778.0 6  00  AA  X  X0
779.0 6  00  AA  X  X0
780.0 6  00  AA  X  X0
781.0 6  00  AA  X  80
782.0 6  00  AA  X  X0
784.0 6  00  AA  X  00
787.0 6  00  AA  5  00 <--
800.0 6  80  AA  5  00
808.0 6  80  AA  X  X0
810.0 6  80  AA  X  80
811.0 6  80  AA  X  80
812.0 6  80  AA  X  80
813.0 6  80  AA  8  80 <--

TIME  ^func  ^b     ^out
        ^a      ^psw
```

Appendix E:

During the course of the design some circuit parts were desired that did not exist in The Library. One of these parts was the TRILAT (Tri-State Latch). This part is a combination of a one bit latch and a tristate buffer.

The performance of the circuit will be discussed here. However all of the variables present on the schematic and the symbol will not be discussed. For more information on these refer to the *Rochester Institute of Technology's CMOS Standard Cell Library Administrator's Manual* by Larry Rubin et. al.

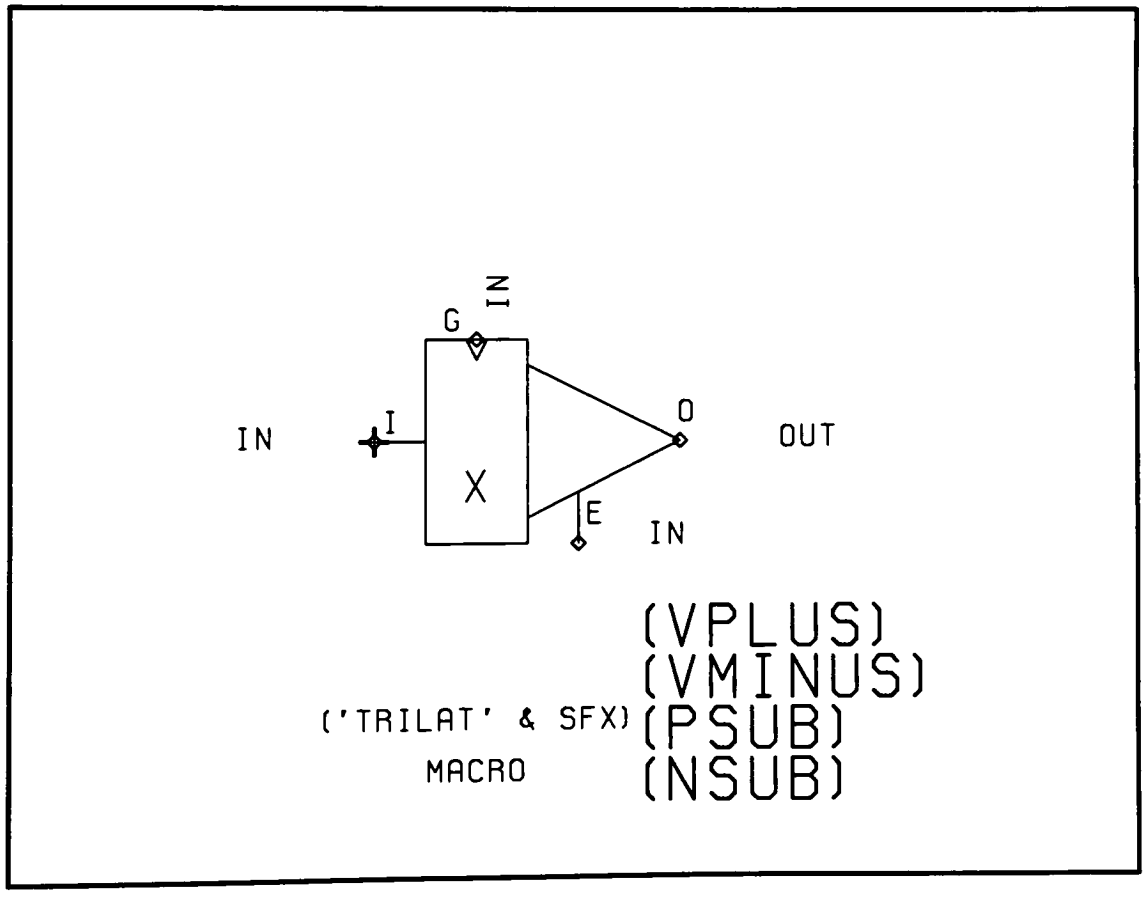

Figure E-1 Logic Symbol of Trilat Circuit

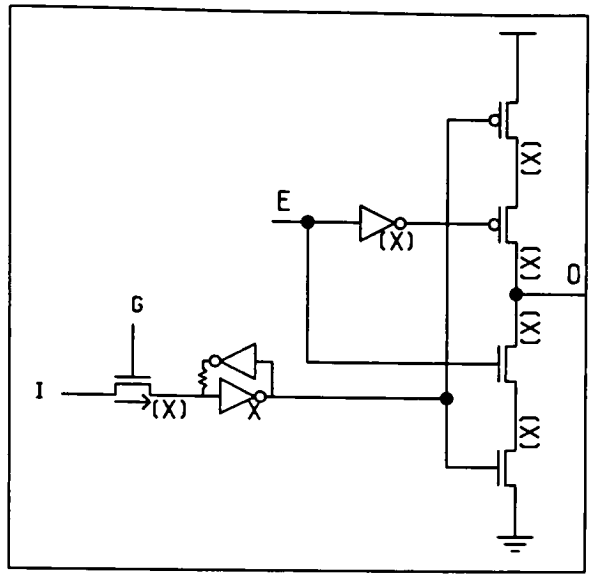

Figure E-2 Transistor Level Schematic of Tri-State Latch for Logic Simulation

The schematic of the circuit is shown in E-2. Various sizes of the circuit were designed, however only the 'C' is used in the microprocessor.

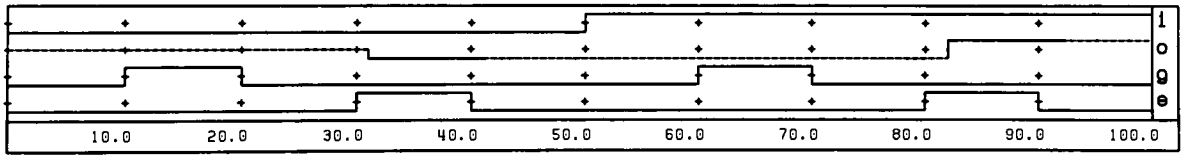

Figure E-3 Logic Simulation Results of Trilat

Above is a captured trace from a Quicksim logical simulation of the Trilat circuit. It can be readily observed that the output of the circuit is in the high impedance condition when the e (enable) signal is low. When the e line is brought high the circuit writes out its contents as a strong signal to the output.

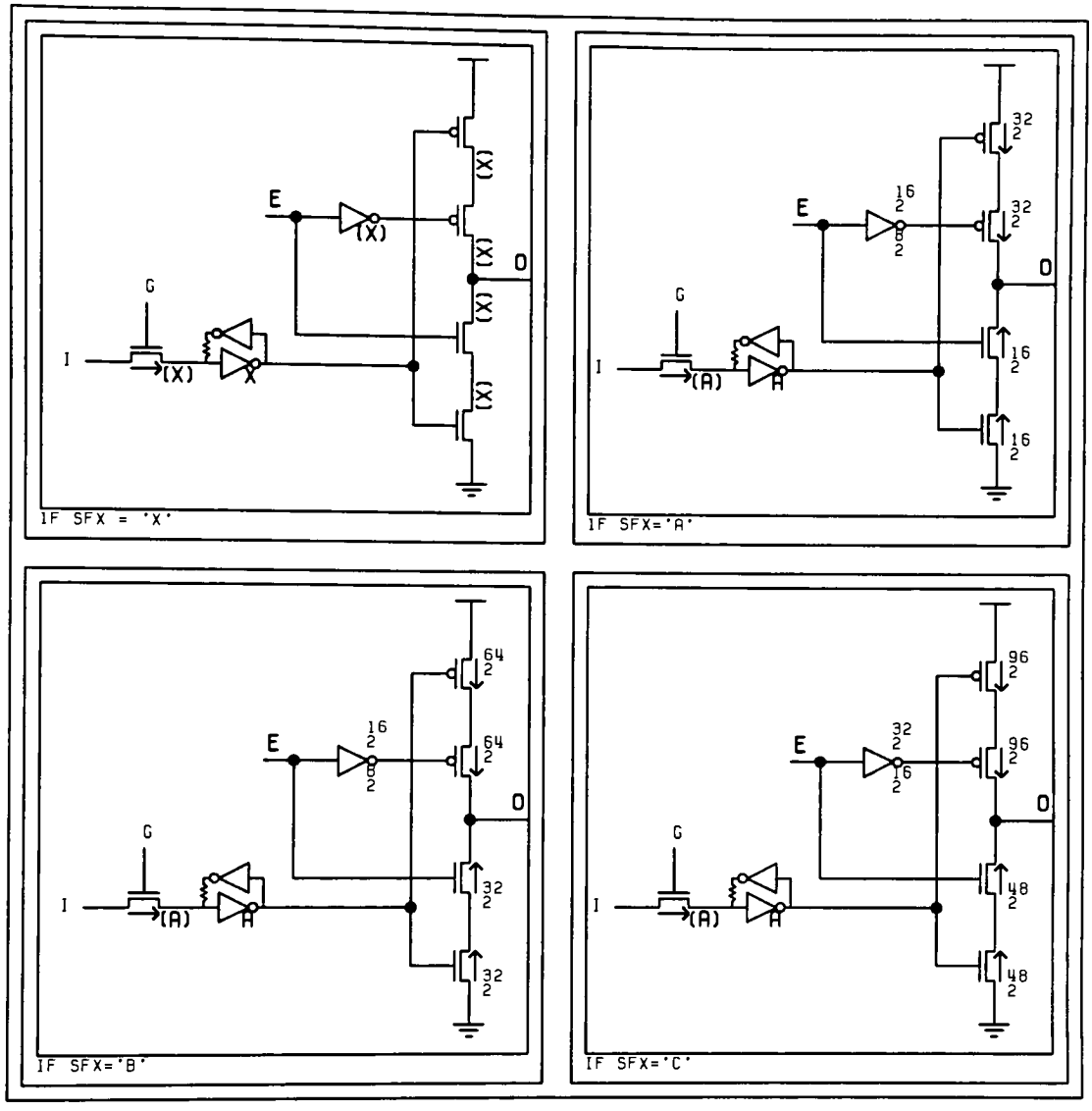

Figure E-4 Transistor Level Schematic of Tri-State Latch for Physical Simulation

A trace of an Accusim physical simulation of the C sized Trilat circuit driving a 0.5 pF capacitive load at 25° C is shown in Figure E-5. The high and low delay times were measured from 50% of maximum input signal to 50% of maximum output signal. Using this technique, the high delay time of the circuit was 3.13 nS and the low delay time was 1.70 nS.

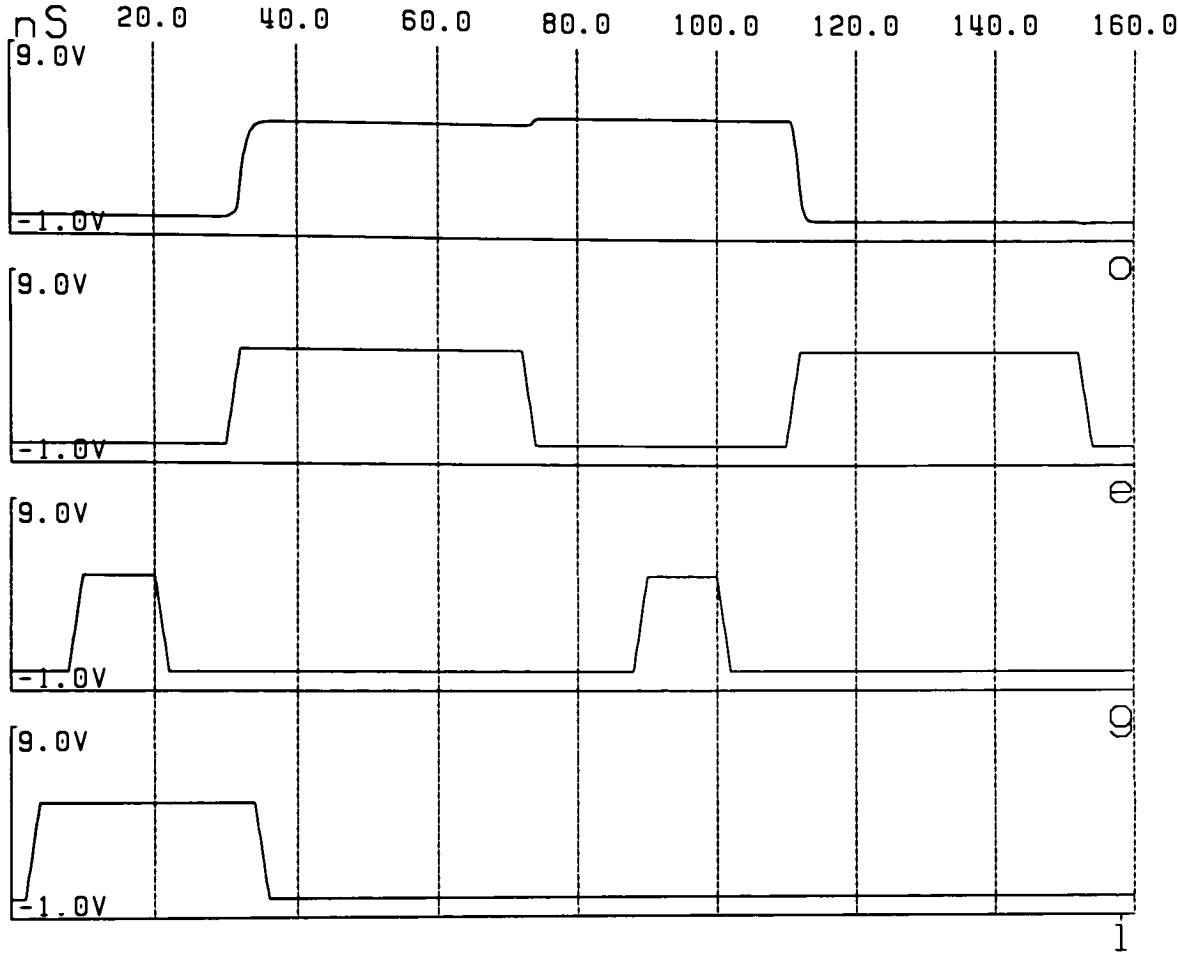

Figure E-5 Simulation Results of Trilat

Appendix F:

The following listings are test programs used in the verification of the processor behavioral and logic circuit models. The programs used are presented first. These listing are in the format of memory address/opcode. Next to the first byte of each instruction is a mnemonic comment of each instruction and the expected result.

These programs listings are only a fraction of the ones used to test, however they are a good representation of the kind of tests performed.

```
#
# This program tests the loading and storing
# instructions for all modes of addressing.
#
00/01;
01/08;  #lda #$25   a<-0x25
02/00;
03/25;
04/10;  #sta #$a0   [$a0]<-0x25
05/00;
06/a0;
07/08;  #ldb  #$a0 b<-$a0
08/10;
09/a0;
0a/10;  #stb #$b0   [$b0]<-a0
0b/10;
0c/b0;
0d/0a;  #ldc $b0    c<-[$b0]<-a0
0e/20;
0f/b0;
10/08;  #ldd #$e0   d<-0xe0
11/30;
12/e0;
13/12;  #std $a0    [$a0]<-e0 = [0x25]<-e0
14/30;
15/a0;
16/0a;  #lda $a0    a<-[$25]<-e0
17/00;
```

```
18/25;
19/00;
1a/00;


#    250.000: /CONTROL_UNIT:: State A
#    500.000: /CONTROL_UNIT:: Reset activated
#    750.000: /CONTROL_UNIT:: State T
#    1250.000: /CONTROL_UNIT:: State T
#    1750.000: /CONTROL_UNIT:: State T
#    2250.000: /CONTROL_UNIT:: State T
#    2750.000: /CONTROL_UNIT:: State T
#    3250.000: /CONTROL_UNIT:: State T
#    3750.000: /CONTROL_UNIT:: State T
#    4250.000: /CONTROL_UNIT:: State T
#    4750.000: /CONTROL_UNIT:: State T
#    5250.000: /CONTROL_UNIT:: State R
#    5500.000: /INSTR_REG:: instr = 0x08
#    5500.000: /INSTR_REG:: instr(2:0)=0x01
#    5500.000: /INSTR_REG:: decode type = 0x00
#    5750.000: /INSTR_REG:: instr = 0x08
#    5750.000: /INSTR_REG:: instr(2:0)=0x01
#    5750.000: /INSTR_REG:: decode type = 0x00
#    5750.000: /CONTROL_UNIT:: state U
#    6250.000: /CONTROL_UNIT:: State W
#    6750.000: /CONTROL_UNIT:: State W
#    7250.000: /CONTROL_UNIT:: State W
#    7750.000: /CONTROL_UNIT:: State W
#    8250.000: /CONTROL_UNIT:: State W
#    8750.000: /CONTROL_UNIT:: State W
#    9250.000: /MDR:: reading Data=0x00
#    9250.000: /CONTROL_UNIT:: State W
#    9500.000: /MDR:: reading Data=0x00
#    9750.000: /MDR:: reading Data=0x00
#    9750.000: /CONTROL_UNIT:: State Y
#    10250.000: /CONTROL_UNIT:: State Y
#    10750.000: /CONTROL_UNIT:: State Y
#    11250.000: /CONTROL_UNIT:: State V
#    11750.000: /CONTROL_UNIT:: State V
#    12250.000: /CONTROL_UNIT:: State V
#    12750.000: /CONTROL_UNIT:: State V
#    13250.000: /CONTROL_UNIT:: State V
#    13750.000: /CONTROL_UNIT:: State V
#    14250.000: /CONTROL_UNIT:: State V
#    14750.000: /CONTROL_UNIT:: State V
#    15250.000: /CONTROL_UNIT:: State E
#    15255.000: REGA:: reading Data = 0x25
```

```
#    15500.000: REGA:: reading Data = 0x25
#    15750.000: REGA:: reading Data = 0x25
#    15750.000: /CONTROL_UNIT:: State A
#    16250.000: /CONTROL_UNIT:: State A
#    16750.000: /CONTROL_UNIT:: State A
#    17250.000: /CONTROL_UNIT:: State Q
#    17750.000: /CONTROL_UNIT:: State Q
#    18250.000: /CONTROL_UNIT:: State Q
#    18750.000: /CONTROL_UNIT:: State Q
#    19250.000: /CONTROL_UNIT:: State Q
#    19750.000: /CONTROL_UNIT:: State Q
#    20250.000: /CONTROL_UNIT:: State Q
#    20750.000: /CONTROL_UNIT:: State R
#    21000.000: /INSTR_REG:: instr = 0x10
#    21000.000: /INSTR_REG:: instr(2:0)=0x02
#    21000.000: /INSTR_REG:: decode type = 0x00
#    21250.000: /INSTR_REG:: instr = 0x10
#    21250.000: /INSTR_REG:: instr(2:0)=0x02
#    21250.000: /INSTR_REG:: decode type = 0x00
#    21250.000: /CONTROL_UNIT:: state U
#    21750.000: /CONTROL_UNIT:: State W
#    21755.000: /MDR:: reading Data=0x10
#    22000.000: /MDR:: reading Data=0x10
#    22250.000: /CONTROL_UNIT:: State W
#    22750.000: /CONTROL_UNIT:: State W
#    23250.000: /CONTROL_UNIT:: State W
#    23750.000: /CONTROL_UNIT:: State W
#    24250.000: /CONTROL_UNIT:: State W
#    24750.000: /CONTROL_UNIT:: State W
#    25250.000: /MDR:: reading Data=0x00
#    25250.000: /CONTROL_UNIT:: State W
#    25500.000: /MDR:: reading Data=0x00
#    25750.000: /MDR:: reading Data=0x00
#    25750.000: /CONTROL_UNIT:: State Y
#    26250.000: /CONTROL_UNIT:: State Y
#    26750.000: /CONTROL_UNIT:: State Y
#    27250.000: /CONTROL_UNIT:: State V
#    27750.000: /CONTROL_UNIT:: State V
#    28250.000: /CONTROL_UNIT:: State V
#    28750.000: /CONTROL_UNIT:: State V
#    29250.000: /CONTROL_UNIT:: State V
#    29750.000: /CONTROL_UNIT:: State V
#    30250.000: /CONTROL_UNIT:: State V
#    30750.000: /CONTROL_UNIT:: State V
#    31250.000: /CONTROL_UNIT:: State D
#    31255.000: /MDR:: reading Data=0xa0
#    31500.000: /MDR:: reading Data=0xa0
```

```
#   31750.000: /MDR:: reading Data=0xa0
#   31750.000: /CONTROL_UNIT:: State Z1
#   32250.000: /CONTROL_UNIT:: State Z1
#   32750.000: /CONTROL_UNIT:: State F
#   32755.000: REGA:: Writing 0x25 on bus1
#   32755.000: /MDR:: writing Data=0xa0
#   33000.000: /MDR:: writing Data=0xa0
#   33000.000: REGA:: Writing 0x25 on bus1
#   33250.000: /MDR:: writing Data=0xa0
#   33250.000: REGA:: Writing 0x25 on bus1
#   33250.000: /CONTROL_UNIT:: State F
#   33500.000: /MDR:: writing Data=0xa0
#   33500.000: REGA:: Writing 0x25 on bus1
#   33750.000: /MDR:: writing Data=0xa0
#   33750.000: REGA:: Writing 0x25 on bus1
#   33750.000: /CONTROL_UNIT:: State F
#   34000.000: /MDR:: writing Data=0xa0
#   34000.000: REGA:: Writing 0x25 on bus1
#   34250.000: /MDR:: writing Data=0xa0
#   34250.000: REGA:: Writing 0x25 on bus1
#   34250.000: /CONTROL_UNIT:: State F
#   34500.000: /MDR:: writing Data=0xa0
#   34500.000: REGA:: Writing 0x25 on bus1
#   34750.000: /MDR:: writing Data=0xa0
#   34750.000: REGA:: Writing 0x25 on bus1
#   34750.000: /CONTROL_UNIT:: State F
#   35000.000: /MDR:: writing Data=0xa0
#   35000.000: REGA:: Writing 0x25 on bus1
#   35250.000: /MDR:: writing Data=0xa0
#   35250.000: REGA:: Writing 0x25 on bus1
#   35250.000: /CONTROL_UNIT:: State F
#   35500.000: /MDR:: writing Data=0xa0
#   35500.000: REGA:: Writing 0x25 on bus1
#   35750.000: /MDR:: writing Data=0xa0
#   35750.000: REGA:: Writing 0x25 on bus1
#   35750.000: /CONTROL_UNIT:: State A
#   36250.000: /CONTROL_UNIT:: State A
#   36750.000: /CONTROL_UNIT:: State A
#   37250.000: /CONTROL_UNIT:: State Q
#   37750.000: /CONTROL_UNIT:: State Q
#   38250.000: /CONTROL_UNIT:: State Q
#   38750.000: /CONTROL_UNIT:: State Q
#   39250.000: /CONTROL_UNIT:: State Q
#   39750.000: /CONTROL_UNIT:: State Q
#   40250.000: /CONTROL_UNIT:: State Q
#   40750.000: /CONTROL_UNIT:: State R
#   41000.000: /INSTR_REG:: instr = 0x08
```

```
#    41000.000: /INSTR_REG:: instr(2:0)=0x01
#    41000.000: /INSTR_REG:: decode type = 0x00
#    41250.000: /INSTR_REG:: instr = 0x08
#    41250.000: /INSTR_REG:: instr(2:0)=0x01
#    41250.000: /INSTR_REG:: decode type = 0x00
#    41250.000: /CONTROL_UNIT:: state U
#    41750.000: /CONTROL_UNIT:: State W
#    41755.000: /MDR:: reading Data=0x08
#    42000.000: /MDR:: reading Data=0x08
#    42250.000: /CONTROL_UNIT:: State W
#    42750.000: /CONTROL_UNIT:: State W
#    43250.000: /CONTROL_UNIT:: State W
#    43750.000: /CONTROL_UNIT:: State W
#    44250.000: /CONTROL_UNIT:: State W
#    44750.000: /CONTROL_UNIT:: State W
#    45250.000: /MDR:: reading Data=0x10
#    45250.000: /CONTROL_UNIT:: State W
#    45500.000: /MDR:: reading Data=0x10
#    45750.000: /MDR:: reading Data=0x10
#    45750.000: /CONTROL_UNIT:: State Y
#    46250.000: /CONTROL_UNIT:: State Y
#    46750.000: /CONTROL_UNIT:: State Y
#    47250.000: /CONTROL_UNIT:: State V
#    47750.000: /CONTROL_UNIT:: State V
#    48250.000: /CONTROL_UNIT:: State V
#    48750.000: /CONTROL_UNIT:: State V
#    49250.000: /CONTROL_UNIT:: State V
#    49750.000: /CONTROL_UNIT:: State V
#    50250.000: /CONTROL_UNIT:: State V
#    50750.000: /CONTROL_UNIT:: State V
#    51250.000: /CONTROL_UNIT:: State E
#    51255.000: REGB:: Data = 0xa0
#    51500.000: REGB:: Data = 0xa0
#    51750.000: REGB:: Data = 0xa0
#    51750.000: /CONTROL_UNIT:: State A
#    52250.000: /CONTROL_UNIT:: State A
#    52750.000: /CONTROL_UNIT:: State A
#    53250.000: /CONTROL_UNIT:: State Q
#    53750.000: /CONTROL_UNIT:: State Q
#    54250.000: /CONTROL_UNIT:: State Q
#    54750.000: /CONTROL_UNIT:: State Q
#    55250.000: /CONTROL_UNIT:: State Q
#    55750.000: /CONTROL_UNIT:: State Q
#    56250.000: /CONTROL_UNIT:: State Q
#    56750.000: /CONTROL_UNIT:: State R
#    57000.000: /INSTR_REG:: instr = 0x10
#    57000.000: /INSTR_REG:: instr(2:0)=0x02
```

```
#    57000.000: /INSTR_REG:: decode type = 0x00
#    57250.000: /INSTR_REG:: instr = 0x10
#    57250.000: /INSTR_REG:: instr(2:0)=0x02
#    57250.000: /INSTR_REG:: decode type = 0x00
#    57250.000: /CONTROL_UNIT:: state U
#    57750.000: /CONTROL_UNIT:: State W
#    57755.000: /MDR:: reading Data=0x10
#    58000.000: /MDR:: reading Data=0x10
#    58250.000: /CONTROL_UNIT:: State W
#    58750.000: /CONTROL_UNIT:: State W
#    59250.000: /CONTROL_UNIT:: State W
#    59750.000: /CONTROL_UNIT:: State W
#    60250.000: /CONTROL_UNIT:: State W
#    60750.000: /CONTROL_UNIT:: State W
#    61250.000: /MDR:: reading Data=0x10
#    61250.000: /CONTROL_UNIT:: State W
#    61500.000: /MDR:: reading Data=0x10
#    61750.000: /MDR:: reading Data=0x10
#    61750.000: /CONTROL_UNIT:: State Y
#    62250.000: /CONTROL_UNIT:: State Y
#    62750.000: /CONTROL_UNIT:: State Y
#    63250.000: /CONTROL_UNIT:: State V
#    63750.000: /CONTROL_UNIT:: State V
#    64250.000: /CONTROL_UNIT:: State V
#    64750.000: /CONTROL_UNIT:: State V
#    65250.000: /CONTROL_UNIT:: State V
#    65750.000: /CONTROL_UNIT:: State V
#    66250.000: /CONTROL_UNIT:: State V
#    66750.000: /CONTROL_UNIT:: State V
#    67250.000: /CONTROL_UNIT:: State D
#    67255.000: /MDR:: reading Data=0xb0
#    67500.000: /MDR:: reading Data=0xb0
#    67750.000: /MDR:: reading Data=0xb0
#    67750.000: /CONTROL_UNIT:: State Z1
#    68250.000: /CONTROL_UNIT:: State Z1
#    68750.000: /CONTROL_UNIT:: State F
#    68755.000: REGB:: Writing 0xa0 on bus1
#    68755.000: /MDR:: writing Data=0xb0
#    69000.000: /MDR:: writing Data=0xb0
#    69000.000: REGB:: Writing 0xa0 on bus1
#    69250.000: /MDR:: writing Data=0xb0
#    69250.000: REGB:: Writing 0xa0 on bus1
#    69250.000: /CONTROL_UNIT:: State F
#    69500.000: /MDR:: writing Data=0xb0
#    69500.000: REGB:: Writing 0xa0 on bus1
#    69750.000: /MDR:: writing Data=0xb0
#    69750.000: REGB:: Writing 0xa0 on bus1
```

```
#    69750.000: /CONTROL_UNIT:: State F
#    70000.000: /MDR:: writing Data=0xb0
#    70000.000: REGB:: Writing 0xa0 on bus1
#    70250.000: /MDR:: writing Data=0xb0
#    70250.000: REGB:: Writing 0xa0 on bus1
#    70250.000: /CONTROL_UNIT:: State F
#    70500.000: /MDR:: writing Data=0xb0
#    70500.000: REGB:: Writing 0xa0 on bus1
#    70750.000: /MDR:: writing Data=0xb0
#    70750.000: REGB:: Writing 0xa0 on bus1
#    70750.000: /CONTROL_UNIT:: State F
#    71000.000: /MDR:: writing Data=0xb0
#    71000.000: REGB:: Writing 0xa0 on bus1
#    71250.000: /MDR:: writing Data=0xb0
#    71250.000: REGB:: Writing 0xa0 on bus1
#    71250.000: /CONTROL_UNIT:: State F
#    71500.000: /MDR:: writing Data=0xb0
#    71500.000: REGB:: Writing 0xa0 on bus1
#    71750.000: /MDR:: writing Data=0xb0
#    71750.000: REGB:: Writing 0xa0 on bus1
#    71750.000: /CONTROL_UNIT:: State A
#    72250.000: /CONTROL_UNIT:: State A
#    72750.000: /CONTROL_UNIT:: State A
#    73250.000: /CONTROL_UNIT:: State Q
#    73750.000: /CONTROL_UNIT:: State Q
#    74250.000: /CONTROL_UNIT:: State Q
#    74750.000: /CONTROL_UNIT:: State Q
#    75250.000: /CONTROL_UNIT:: State Q
#    75750.000: /CONTROL_UNIT:: State Q
#    76250.000: /CONTROL_UNIT:: State Q
#    76750.000: /CONTROL_UNIT:: State R
#    77000.000: /INSTR_REG:: instr = 0x0a
#    77000.000: /INSTR_REG:: instr(2:0)=0x01
#    77000.000: /INSTR_REG:: decode type = 0x00
#    77250.000: /INSTR_REG:: instr = 0x0a
#    77250.000: /INSTR_REG:: instr(2:0)=0x01
#    77250.000: /INSTR_REG:: decode type = 0x00
#    77250.000: /CONTROL_UNIT:: state U
#    77750.000: /CONTROL_UNIT:: State W
#    77755.000: /MDR:: reading Data=0x0a
#    78000.000: /MDR:: reading Data=0x0a
#    78250.000: /CONTROL_UNIT:: State W
#    78750.000: /CONTROL_UNIT:: State W
#    79250.000: /CONTROL_UNIT:: State W
#    79750.000: /CONTROL_UNIT:: State W
#    80250.000: /CONTROL_UNIT:: State W
#    80750.000: /CONTROL_UNIT:: State W
```

.

```
#   81250.000: /MDR:: reading Data=0x20
#   81250.000: /CONTROL_UNIT:: State W
#   81500.000: /MDR:: reading Data=0x20
#   81750.000: /MDR:: reading Data=0x20
#   81750.000: /CONTROL_UNIT:: State Y
#   82250.000: /CONTROL_UNIT:: State Y
#   82750.000: /CONTROL_UNIT:: State Y
#   83250.000: /CONTROL_UNIT:: State C
#   83750.000: /CONTROL_UNIT:: State C
#   84250.000: /CONTROL_UNIT:: State C
#   84750.000: /CONTROL_UNIT:: State C
#   85250.000: /CONTROL_UNIT:: State C
#   85750.000: /CONTROL_UNIT:: State C
#   86250.000: /CONTROL_UNIT:: State C
#   86750.000: /CONTROL_UNIT:: State C
#   87250.000: /CONTROL_UNIT:: State C
#   87750.000: /CONTROL_UNIT:: State C
#   88250.000: /CONTROL_UNIT:: State C
#   88750.000: /CONTROL_UNIT:: State E
#   88755.000: REGC:: reading Data = 0xa0
#   89000.000: REGC:: reading Data = 0xa0
#   89250.000: REGC:: reading Data = 0xa0
#   89250.000: /CONTROL_UNIT:: State A
#   89750.000: /CONTROL_UNIT:: State A
#   90250.000: /CONTROL_UNIT:: State A
#   90750.000: /CONTROL_UNIT:: State Q
#   91250.000: /CONTROL_UNIT:: State Q
#   91750.000: /CONTROL_UNIT:: State Q
#   92250.000: /CONTROL_UNIT:: State Q
#   92750.000: /CONTROL_UNIT:: State Q
#   93250.000: /CONTROL_UNIT:: State Q
#   93750.000: /CONTROL_UNIT:: State Q
#   94250.000: /CONTROL_UNIT:: State R
#   94500.000: /INSTR_REG:: instr = 0x08
#   94500.000: /INSTR_REG:: instr(2:0)=0x01
#   94500.000: /INSTR_REG:: decode type = 0x00
#   94750.000: /INSTR_REG:: instr = 0x08
#   94750.000: /INSTR_REG:: instr(2:0)=0x01
#   94750.000: /INSTR_REG:: decode type = 0x00
#   94750.000: /CONTROL_UNIT:: state U
#   95250.000: /CONTROL_UNIT:: State W
#   95255.000: /MDR:: reading Data=0x08
#   95500.000: /MDR:: reading Data=0x08
#   95750.000: /CONTROL_UNIT:: State W
#   96250.000: /CONTROL_UNIT:: State W
#   96750.000: /CONTROL_UNIT:: State W
#   97250.000: /CONTROL_UNIT:: State W
```

```
#   97750.000: /CONTROL_UNIT:: State W
#   98250.000: /CONTROL_UNIT:: State W
#   98750.000: /MDR:: reading Data=0x30
#   98750.000: /CONTROL_UNIT:: State W
#   99000.000: /MDR:: reading Data=0x30
#   99250.000: /MDR:: reading Data=0x30
#   99250.000: /CONTROL_UNIT:: State Y
#   99750.000: /CONTROL_UNIT:: State Y
#   100250.000: /CONTROL_UNIT:: State Y
#   100750.000: /CONTROL_UNIT:: State V
#   101250.000: /CONTROL_UNIT:: State V
#   101750.000: /CONTROL_UNIT:: State V
#   102250.000: /CONTROL_UNIT:: State V
#   102750.000: /CONTROL_UNIT:: State V
#   103250.000: /CONTROL_UNIT:: State V
#   103750.000: /CONTROL_UNIT:: State V
#   104250.000: /CONTROL_UNIT:: State V
#   104750.000: /CONTROL_UNIT:: State E
#   104755.000: REGD:: Data = 0xe0
#   105000.000: REGD:: Data = 0xe0
#   105250.000: REGD:: Data = 0xe0
#   105250.000: /CONTROL_UNIT:: State A
#   105750.000: /CONTROL_UNIT:: State A
#   106250.000: /CONTROL_UNIT:: State A
#   106750.000: /CONTROL_UNIT:: State Q
#   107250.000: /CONTROL_UNIT:: State Q
#   107750.000: /CONTROL_UNIT:: State Q
#   108250.000: /CONTROL_UNIT:: State Q
#   108750.000: /CONTROL_UNIT:: State Q
#   109250.000: /CONTROL_UNIT:: State Q
#   109750.000: /CONTROL_UNIT:: State Q
#   110250.000: /CONTROL_UNIT:: State R
#   110500.000: /INSTR_REG:: instr = 0x12
#   110500.000: /INSTR_REG:: instr(2:0)=0x02
#   110500.000: /INSTR_REG:: decode type = 0x00
#   110750.000: /INSTR_REG:: instr = 0x12
#   110750.000: /INSTR_REG:: instr(2:0)=0x02
#   110750.000: /INSTR_REG:: decode type = 0x00
#   110750.000: /CONTROL_UNIT:: state U
#   111250.000: /CONTROL_UNIT:: State W
#   111255.000: /MDR:: reading Data=0x12
#   111500.000: /MDR:: reading Data=0x12
#   111750.000: /CONTROL_UNIT:: State W
#   112250.000: /CONTROL_UNIT:: State W
#   112750.000: /CONTROL_UNIT:: State W
#   113250.000: /CONTROL_UNIT:: State W
#   113750.000: /CONTROL_UNIT:: State W
```

```
#   114250.000: /CONTROL_UNIT:: State W
#   114750.000: /MDR:: reading Data=0x30
#   114750.000: /CONTROL_UNIT:: State W
#   115000.000: /MDR:: reading Data=0x30
#   115250.000: /MDR:: reading Data=0x30
#   115250.000: /CONTROL_UNIT:: State Y
#   115750.000: /CONTROL_UNIT:: State Y
#   116250.000: /CONTROL_UNIT:: State Y
#   116750.000: /CONTROL_UNIT:: State C
#   117250.000: /CONTROL_UNIT:: State C
#   117750.000: /CONTROL_UNIT:: State C
#   118250.000: /CONTROL_UNIT:: State C
#   118750.000: /CONTROL_UNIT:: State C
#   119250.000: /CONTROL_UNIT:: State C
#   119750.000: /CONTROL_UNIT:: State C
#   120250.000: /CONTROL_UNIT:: State C
#   120750.000: /CONTROL_UNIT:: State C
#   121250.000: /CONTROL_UNIT:: State C
#   121750.000: /CONTROL_UNIT:: State C
#   122250.000: /CONTROL_UNIT:: State D
#   122255.000: /MDR:: reading Data=0x25
#   122500.000: /MDR:: reading Data=0x25
#   122750.000: /MDR:: reading Data=0x25
#   122750.000: /CONTROL_UNIT:: State Z1
#   123250.000: /CONTROL_UNIT:: State Z1
#   123750.000: /CONTROL_UNIT:: State F
#   123755.000: REGD:: Writing 0xe0 on bus1
#   123755.000: /MDR:: writing Data=0x25
#   124000.000: /MDR:: writing Data=0x25
#   124000.000: REGD:: Writing 0xe0 on bus1
#   124250.000: /MDR:: writing Data=0x25
#   124250.000: REGD:: Writing 0xe0 on bus1
#   124250.000: /CONTROL_UNIT:: State F
#   124500.000: /MDR:: writing Data=0x25
#   124500.000: REGD:: Writing 0xe0 on bus1
#   124750.000: /MDR:: writing Data=0x25
#   124750.000: REGD:: Writing 0xe0 on bus1
#   124750.000: /CONTROL_UNIT:: State F
#   125000.000: /MDR:: writing Data=0x25
#   125000.000: REGD:: Writing 0xe0 on bus1
#   125250.000: /MDR:: writing Data=0x25
#   125250.000: REGD:: Writing 0xe0 on bus1
#   125250.000: /CONTROL_UNIT:: State F
#   125500.000: /MDR:: writing Data=0x25
#   125500.000: REGD:: Writing 0xe0 on bus1
#   125750.000: /MDR:: writing Data=0x25
#   125750.000: REGD:: Writing 0xe0 on bus1
```

```
#   125750.000: /CONTROL_UNIT:: State F
#   126000.000: /MDR:: writing Data=0x25
#   126000.000: REGD:: Writing 0xe0 on bus1
#   126250.000: /MDR:: writing Data=0x25
#   126250.000: REGD:: Writing 0xe0 on bus1
#   126250.000: /CONTROL_UNIT:: State F
#   126500.000: /MDR:: writing Data=0x25
#   126500.000: REGD:: Writing 0xe0 on bus1
#   126750.000: /MDR:: writing Data=0x25
#   126750.000: REGD:: Writing 0xe0 on bus1
#   126750.000: /CONTROL_UNIT:: State F
#   127000.000: /MDR:: writing Data=0x25
#   127000.000: REGD:: Writing 0xe0 on bus1
#   127250.000: /MDR:: writing Data=0x25
#   127250.000: REGD:: Writing 0xe0 on bus1
#   127250.000: /CONTROL_UNIT:: State F
#   127500.000: /MDR:: writing Data=0x25
#   127500.000: REGD:: Writing 0xe0 on bus1
#   127750.000: /MDR:: writing Data=0x25
#   127750.000: REGD:: Writing 0xe0 on bus1
#   127750.000: /CONTROL_UNIT:: State F
#   128000.000: /MDR:: writing Data=0x25
#   128000.000: REGD:: Writing 0xe0 on bus1
#   128250.000: /MDR:: writing Data=0x25
#   128250.000: REGD:: Writing 0xe0 on bus1
#   128250.000: /CONTROL_UNIT:: State F
#   128500.000: /MDR:: writing Data=0x25
#   128500.000: REGD:: Writing 0xe0 on bus1
#   128750.000: /MDR:: writing Data=0x25
#   128750.000: REGD:: Writing 0xe0 on bus1
#   128750.000: /CONTROL_UNIT:: State F
#   129000.000: /MDR:: writing Data=0x25
#   129000.000: REGD:: Writing 0xe0 on bus1
#   129250.000: /MDR:: writing Data=0x25
#   129250.000: REGD:: Writing 0xe0 on bus1
#   129250.000: /CONTROL_UNIT:: State A
#   129750.000: /CONTROL_UNIT:: State A
#   130250.000: /CONTROL_UNIT:: State A
#   130750.000: /CONTROL_UNIT:: State Q
#   131250.000: /CONTROL_UNIT:: State Q
#   131750.000: /CONTROL_UNIT:: State Q
#   132250.000: /CONTROL_UNIT:: State Q
#   132750.000: /CONTROL_UNIT:: State Q
#   133250.000: /CONTROL_UNIT:: State Q
#   133750.000: /CONTROL_UNIT:: State Q
#   134250.000: /CONTROL_UNIT:: State R
#   134500.000: /INSTR_REG:: instr = 0x0a
```

```
#    134500.000: /INSTR_REG:: instr(2:0)=0x01
#    134500.000: /INSTR_REG:: decode type = 0x00
#    134750.000: /INSTR_REG:: instr = 0x0a
#    134750.000: /INSTR_REG:: instr(2:0)=0x01
#    134750.000: /INSTR_REG:: decode type = 0x00
#    134750.000: /CONTROL_UNIT:: state U
#    135250.000: /CONTROL_UNIT:: State W
#    135255.000: /MDR:: reading Data=0x0a
#    135500.000: /MDR:: reading Data=0x0a
#    135750.000: /CONTROL_UNIT:: State W
#    136250.000: /MDR:: reading Data=0x00
#    136250.000: /CONTROL_UNIT:: State W
#    136500.000: /MDR:: reading Data=0x00
#    136750.000: /MDR:: reading Data=0x00
#    136750.000: /CONTROL_UNIT:: State Y
#    137250.000: /CONTROL_UNIT:: State Y
#    137750.000: /CONTROL_UNIT:: State Y
#    138250.000: /CONTROL_UNIT:: State C
#    138750.000: /CONTROL_UNIT:: State C
#    139250.000: /CONTROL_UNIT:: State C
#    139750.000: /CONTROL_UNIT:: State C
#    140250.000: /CONTROL_UNIT:: State C
#    140750.000: /CONTROL_UNIT:: State C
#    141250.000: /CONTROL_UNIT:: State C
#    141750.000: /CONTROL_UNIT:: State C
#    142250.000: /CONTROL_UNIT:: State C
#    142750.000: /CONTROL_UNIT:: State C
#    143250.000: /CONTROL_UNIT:: State C
#    143750.000: /CONTROL_UNIT:: State E
#    143755.000: REGA:: reading Data = 0xe0
#    144000.000: REGA:: reading Data = 0xe0
#    144250.000: REGA:: reading Data = 0xe0
#    144250.000: /CONTROL_UNIT:: State A
#    144750.000: /CONTROL_UNIT:: State A
#    145250.000: /CONTROL_UNIT:: State A
#    145750.000: /CONTROL_UNIT:: State Q
#    146250.000: /CONTROL_UNIT:: State Q
#    146750.000: /CONTROL_UNIT:: State Q
#    147250.000: /CONTROL_UNIT:: State Q
#    147750.000: /CONTROL_UNIT:: State Q
#    148250.000: /CONTROL_UNIT:: State Q
#    148750.000: /CONTROL_UNIT:: State Q
#    149250.000: /CONTROL_UNIT:: State R
#    149500.000: /INSTR_REG:: instr = 0x00
#    149500.000: /INSTR_REG:: instr(2:0)=0x00
#    149500.000: /INSTR_REG:: decode type = 0x00
#    149750.000: /INSTR_REG:: instr = 0x00
```

```
#    149750.000: /INSTR_REG:: instr(2:0)=0x00
#    149750.000: /INSTR_REG:: decode type = 0x00
#    149750.000: /CONTROL_UNIT:: state U
#    149750.000: /CONTROL_UNIT:: Nop..next state = A
#    150250.000: /CONTROL_UNIT:: State A
#    150750.000: /CONTROL_UNIT:: State Q
```

```
#
#  This program tests the compare instruction
#
#  The A and B registers are loaded with numbers
#  and then the compare is performed.  The results
#  are placed in the Flags Register.
#
00/01;
01/08; #ld a, #$05
02/00;
03/05;
04/08; #ld b, #$10
05/10;
06/10;
07/7c; #cmp a,b  PSW=0x08
08/01;
09/7c; #cmp b,a  PSW=0x00
0a/10;
0b/78; #cmp a,#$05  PSW=0x04
0c/0f;
0d/05;
0e/00;
0f/00;


#   500.000: /BUS_UNIT:: Reset Activated
#   500.000: /BUS_UNIT:: State C
#   1000.000: /BUS_UNIT:: State C
#   1500.000: /BUS_UNIT:: state G
#   2000.000: /BUS_UNIT:: state P
#   2500.000: /BUS_UNIT:: state I
#   3000.000: /BUS_UNIT:: state O
#   3500.000: /BUS_UNIT:: state L
#   4000.000: /BUS_UNIT:: state D
#   4500.000: /BUS_UNIT:: state N
#   5000.000: /BUS_UNIT:: state N
#   5500.000: /INSTR_REG:: instr = 0x08
#   5500.000: /INSTR_REG:: instr(2:0)=0x01
#   5500.000: /INSTR_REG:: decode type = 0x00
#   5500.000: /BUS_UNIT:: state N
#   5750.000: /INSTR_REG:: instr = 0x08
#   5750.000: /INSTR_REG:: instr(2:0)=0x01
#   5750.000: /INSTR_REG:: decode type = 0x00
#   6000.000: /BUS_UNIT:: State A
#   6000.000: /BUS_UNIT:: Pre-fetch empty...
#   6000.000: /BUS_UNIT:: Fill the prefetch
```

```
#    6500.000: /BUS_UNIT:: State C
#    6750.000: /MDR:: reading Data=0xff
#    7000.000: /MDR:: reading Data=0xff
#    7000.000: /BUS_UNIT:: State C
#    7500.000: /BUS_UNIT:: state G
#    8000.000: /BUS_UNIT:: state P
#    8500.000: /BUS_UNIT:: state H
#    9000.000: /BUS_UNIT:: state J
#    9250.000: /MDR:: reading Data=0x00
#    9500.000: /MDR:: reading Data=0x00
#    9500.000: /BUS_UNIT:: state J
#    9750.000: /MDR:: reading Data=0x00
#    10000.000: /BUS_UNIT:: state J
#    10500.000: /BUS_UNIT:: state F
#    11000.000: /BUS_UNIT:: State A
#    11000.000: /BUS_UNIT:: Pre-fetch empty...
#    11000.000: /BUS_UNIT:: Fill the prefetch
#    11500.000: /BUS_UNIT:: State C
#    12000.000: /BUS_UNIT:: State C
#    12500.000: /BUS_UNIT:: state G
#    13000.000: /BUS_UNIT:: state P
#    13500.000: /BUS_UNIT:: state H
#    14000.000: /BUS_UNIT:: State A
#    14000.000: /BUS_UNIT:: Q[2] = 0x05

#    14500.000: /BUS_UNIT:: state J
#    15000.000: /BUS_UNIT:: state J
#    15255.000: REGA:: reading Data = 0x05
#    15500.000: REGA:: reading Data = 0x05
#    15500.000: /BUS_UNIT:: state J
#    15750.000: REGA:: reading Data = 0x05
#    16000.000: /BUS_UNIT:: state J
#    16500.000: /BUS_UNIT:: state F
#    17000.000: /BUS_UNIT:: State A
#    17000.000: /BUS_UNIT:: Pre-fetch empty...
#    17000.000: /BUS_UNIT:: Fill the prefetch
#    17500.000: /BUS_UNIT:: State C
#    18000.000: /BUS_UNIT:: State C
#    18500.000: /BUS_UNIT:: state G
#    19000.000: /BUS_UNIT:: state P
#    19500.000: /BUS_UNIT:: state H
#    20000.000: /BUS_UNIT:: state J
#    20500.000: /BUS_UNIT:: state J
#    21000.000: /INSTR_REG:: instr = 0x08
#    21000.000: /INSTR_REG:: instr(2:0)=0x01
#    21000.000: /INSTR_REG:: decode type = 0x00
#    21000.000: /BUS_UNIT:: state J
```

```
#    21250.000: /INSTR_REG:: instr = 0x08
#    21250.000: /INSTR_REG:: instr(2:0)=0x01
#    21250.000: /INSTR_REG:: decode type = 0x00
#    21500.000: /BUS_UNIT:: state F
#    21755.000: /MDR:: reading Data=0x08
#    22000.000: /MDR:: reading Data=0x08
#    22000.000: /BUS_UNIT:: State A
#    22000.000: /BUS_UNIT:: Pre-fetch empty...
#    22500.000: /BUS_UNIT:: State C
#    23000.000: /BUS_UNIT:: State C
#    23500.000: /BUS_UNIT:: state G
#    24000.000: /BUS_UNIT:: state P
#    24500.000: /BUS_UNIT:: state H
#    25000.000: /BUS_UNIT:: state J
#    25250.000: /MDR:: reading Data=0x10
#    25500.000: /MDR:: reading Data=0x10
#    25500.000: /BUS_UNIT:: state J
#    25750.000: /MDR:: reading Data=0x10
#    26000.000: /BUS_UNIT:: state J
#    26500.000: /BUS_UNIT:: state F
#    27000.000: /BUS_UNIT:: State A
#    27000.000: /BUS_UNIT:: Pre-fetch empty...
#    27000.000: /BUS_UNIT:: Fill the prefetch
#    27500.000: /BUS_UNIT:: State C
#    28000.000: /BUS_UNIT:: State C
#    28500.000: /BUS_UNIT:: state G
#    29000.000: /BUS_UNIT:: state P
#    29500.000: /BUS_UNIT:: state H
#    30000.000: /BUS_UNIT:: State A
#    30000.000: /BUS_UNIT:: Q[1] = 0x10

#    30500.000: /BUS_UNIT:: state J
#    31000.000: /BUS_UNIT:: state J
#    31255.000: REGB:: Data = 0x10
#    31500.000: REGB:: Data = 0x10
#    31500.000: /BUS_UNIT:: state J
#    31750.000: REGB:: Data = 0x10
#    32000.000: /BUS_UNIT:: state J
#    32500.000: /BUS_UNIT:: state F
#    33000.000: /BUS_UNIT:: State A
#    33000.000: /BUS_UNIT:: Pre-fetch empty...
#    33000.000: /BUS_UNIT:: Fill the prefetch
#    33500.000: /BUS_UNIT:: State C
#    34000.000: /BUS_UNIT:: State C
#    34500.000: /BUS_UNIT:: state G
#    35000.000: /BUS_UNIT:: state P
#    35500.000: /BUS_UNIT:: state H
```

```
#    36000.000: /BUS_UNIT:: state J
#    36500.000: /BUS_UNIT:: state J
#    37000.000: /INSTR_REG:: instr = 0x7c
#    37000.000: /INSTR_REG:: instr(2:0)=0x07
#    37000.000: /INSTR_REG:: decode type = 0x01
#    37000.000: /BUS_UNIT:: state J
#    37250.000: /INSTR_REG:: instr = 0x7c
#    37250.000: /INSTR_REG:: instr(2:0)=0x07
#    37250.000: /INSTR_REG:: decode type = 0x01
#    37500.000: /BUS_UNIT:: state F
#    37755.000: /MDR:: reading Data=0x7c
#    38000.000: /MDR:: reading Data=0x7c
#    38000.000: /BUS_UNIT:: State A
#    38000.000: /BUS_UNIT:: Pre-fetch empty...
#    38500.000: /BUS_UNIT:: State C
#    39000.000: /BUS_UNIT:: State C
#    39500.000: /BUS_UNIT:: state G
#    40000.000: /BUS_UNIT:: state P
#    40500.000: /BUS_UNIT:: state H
#    41000.000: /BUS_UNIT:: state J
#    41250.000: /MDR:: reading Data=0x01
#    41500.000: /MDR:: reading Data=0x01
#    41500.000: /BUS_UNIT:: state J
#    41750.000: /MDR:: reading Data=0x01
#    42000.000: /BUS_UNIT:: state J
#    42500.000: /BUS_UNIT:: state F
#    43000.000: /BUS_UNIT:: State A
#    43000.000: /BUS_UNIT:: Pre-fetch empty...
#    43000.000: /BUS_UNIT:: Fill the prefetch
#    43255.000: REGA:: Writing 0x05 on bus1
#    43255.000: REGB:: Writing 0x10 on bus0
#    43500.000: /FLAGS_REG:: Just took i0 as 08
#    43500.000: REGA:: Writing 0x05 on bus1
#    43500.000: REGB:: Writing 0x10 on bus0
#    43500.000: /BUS_UNIT:: State C
#    43750.000: /FLAGS_REG:: Just took i0 as 08
#    43750.000: REGA:: Writing 0x05 on bus1
#    43750.000: REGB:: Writing 0x10 on bus0
#    44000.000: /BUS_UNIT:: State C
#    44500.000: /BUS_UNIT:: state G
#    45000.000: /BUS_UNIT:: state P
#    45500.000: /BUS_UNIT:: state H
#    46000.000: /BUS_UNIT:: State A
#    46000.000: /BUS_UNIT:: Q[0] = 0x7c

#    46500.000: /BUS_UNIT:: state J
#    47000.000: /BUS_UNIT:: state J
```

```
#    47500.000: /INSTR_REG:: instr = 0x7c
#    47500.000: /INSTR_REG:: instr(2:0)=0x07
#    47500.000: /INSTR_REG:: decode type = 0x01
#    47500.000: /BUS_UNIT:: state J
#    47750.000: /INSTR_REG:: instr = 0x7c
#    47750.000: /INSTR_REG:: instr(2:0)=0x07
#    47750.000: /INSTR_REG:: decode type = 0x01
#    48000.000: /BUS_UNIT:: state F
#    48255.000: /MDR:: reading Data=0x7c
#    48500.000: /MDR:: reading Data=0x7c
#    48500.000: /BUS_UNIT:: State A
#    48500.000: /BUS_UNIT:: Pre-fetch empty...
#    49000.000: /BUS_UNIT:: State C
#    49500.000: /BUS_UNIT:: State C
#    50000.000: /BUS_UNIT:: state G
#    50500.000: /BUS_UNIT:: state P
#    51000.000: /BUS_UNIT:: state H
#    51500.000: /BUS_UNIT:: state J
#    51750.000: /MDR:: reading Data=0x10
#    52000.000: /MDR:: reading Data=0x10
#    52000.000: /BUS_UNIT:: state J
#    52250.000: /MDR:: reading Data=0x10
#    52500.000: /BUS_UNIT:: state J
#    53000.000: /BUS_UNIT:: state F
#    53500.000: /BUS_UNIT:: State A
#    53500.000: /BUS_UNIT:: Pre-fetch empty...
#    53500.000: /BUS_UNIT:: Fill the prefetch
#    53755.000: REGB:: Writing 0x10 on bus1
#    53755.000: REGA:: Writing 0x05 on bus0
#    54000.000: /FLAGS_REG:: Just took i0 as 00
#    54000.000: REGA:: Writing 0x05 on bus0
#    54000.000: REGB:: Writing 0x10 on bus1
#    54000.000: /BUS_UNIT:: State C
#    54250.000: /FLAGS_REG:: Just took i0 as 00
#    54250.000: REGA:: Writing 0x05 on bus0
#    54250.000: REGB:: Writing 0x10 on bus1
#    54500.000: /BUS_UNIT:: State C
#    55000.000: /BUS_UNIT:: state G
#    55500.000: /BUS_UNIT:: state P
#    56000.000: /BUS_UNIT:: state H
#    56500.000: /BUS_UNIT:: State A
#    56500.000: /BUS_UNIT:: Q[2] = 0x78

#    57000.000: /BUS_UNIT:: state J
#    57500.000: /BUS_UNIT:: state J
#    58000.000: /INSTR_REG:: instr = 0x78
#    58000.000: /INSTR_REG:: instr(2:0)=0x07
```

```
#    58000.000: /INSTR_REG:: decode type = 0x01
#    58000.000: /BUS_UNIT:: state J
#    58250.000: /INSTR_REG:: instr = 0x78
#    58250.000: /INSTR_REG:: instr(2:0)=0x07
#    58250.000: /INSTR_REG:: decode type = 0x01
#    58500.000: /BUS_UNIT:: state F
#    58755.000: /MDR:: reading Data=0x78
#    59000.000: /MDR:: reading Data=0x78
#    59000.000: /BUS_UNIT:: State A
#    59000.000: /BUS_UNIT:: Pre-fetch empty...
#    59500.000: /BUS_UNIT:: State C
#    60000.000: /BUS_UNIT:: State C
#    60500.000: /BUS_UNIT:: state G
#    61000.000: /BUS_UNIT:: state P
#    61500.000: /BUS_UNIT:: state H
#    62000.000: /BUS_UNIT:: state J
#    62250.000: /MDR:: reading Data=0x0f
#    62500.000: /MDR:: reading Data=0x0f
#    62500.000: /BUS_UNIT:: state J
#    62750.000: /MDR:: reading Data=0x0f
#    63000.000: /BUS_UNIT:: state J
#    63500.000: /BUS_UNIT:: state F
#    64000.000: /BUS_UNIT:: State A
#    64000.000: /BUS_UNIT:: Pre-fetch empty...
#    64000.000: /BUS_UNIT:: Fill the prefetch
#    64500.000: /BUS_UNIT:: State C
#    65000.000: /BUS_UNIT:: State C
#    65500.000: /BUS_UNIT:: state G
#    66000.000: /BUS_UNIT:: state P
#    66500.000: /BUS_UNIT:: state H
#    67000.000: /BUS_UNIT:: State A
#    67000.000: /BUS_UNIT:: Q[0] = 0x05

#    67500.000: /BUS_UNIT:: state J
#    68000.000: /BUS_UNIT:: state J
#    68255.000: /MDR:: reading Data=0x05
#    68500.000: /MDR:: reading Data=0x05
#    68500.000: /BUS_UNIT:: state J
#    68750.000: /MDR:: reading Data=0x05
#    68755.000: /MDR:: writing Data=0x05
#    68755.000: REGA:: Writing 0x05 on bus1
#    69000.000: /MDR:: writing Data=0x05
#    69000.000: /FLAGS_REG:: Just took i0 as 04
#
# This program is used to test the functionality
# of the shifter.  Also the carry bit in the
# flags register is toggled to vary the carry
```

```
# input to the shifter.
#
00/01;
01/08;  #lda #$01
02/00;
03/01;
04/c8;  #lsla     carry=0, a=#$02,PSW=0x00
05/08;  #ldb #$f1
06/10;
07/f1;
08/c9;  #lslb     carry=1, b=#$e2,PSW=0x09
09/08;  #ldc #$91
0a/20;
0b/91;
0c/d2;  #asrc     carry=1, c=#$c8,PSW=0x09
0d/c2;  #lsrc     carry=0, c=#$64,PSW=0x00
0e/b0;  #scb      carry=1
0f/e0;  #rora     carry=0, a=#$32,PSW=0x00
10/b0;  #scb      carry=1
11/e9;  #rolb     carry=0, b=#$64,PSW=0x00
12/b0;  #scb      carry=1
13/a8;  #ccb      carry=0
14/00;
15/00;
16/00;
17/00;



#    250.000: /CONTROL_UNIT:: State A
#    500.000: /CONTROL_UNIT:: Reset activated
#    500.000: /BUS_UNIT:: Reset Activated
#    500.000: /BUS_UNIT:: State C
#    750.000: /CONTROL_UNIT:: State T
#    1000.000: /BUS_UNIT:: State C
#    1250.000: /CONTROL_UNIT:: State T
#    1500.000: /BUS_UNIT:: state G
#    1750.000: /CONTROL_UNIT:: State T
#    2000.000: /BUS_UNIT:: state P
#    2250.000: /CONTROL_UNIT:: State T
#    2500.000: /BUS_UNIT:: state I
#    2750.000: /CONTROL_UNIT:: State T
#    3000.000: /BUS_UNIT:: state O
#    3250.000: /CONTROL_UNIT:: State T
#    3500.000: /BUS_UNIT:: state L
#    3750.000: /CONTROL_UNIT:: State T
#    4000.000: /BUS_UNIT:: state D
```

```
#    4250.000: /CONTROL_UNIT:: State T
#    4500.000: /BUS_UNIT:: state N
#    4750.000: /CONTROL_UNIT:: State T
#    5000.000: /BUS_UNIT:: state N
#    5250.000: /CONTROL_UNIT:: State R
#    5500.000: /INSTR_REG:: instr = 0x08
#    5500.000: /INSTR_REG:: instr(2:0)=0x01
#    5500.000: /INSTR_REG:: decode type = 0x00
#    5500.000: /BUS_UNIT:: state N
#    5750.000: /INSTR_REG:: instr = 0x08
#    5750.000: /INSTR_REG:: instr(2:0)=0x01
#    5750.000: /INSTR_REG:: decode type = 0x00
#    5750.000: /CONTROL_UNIT:: state U
#    6000.000: /BUS_UNIT:: State A
#    6000.000: /BUS_UNIT:: Pre-fetch empty...
#    6000.000: /BUS_UNIT:: Fill the prefetch
#    6250.000: /CONTROL_UNIT:: State W
#    6750.000: /CONTROL_UNIT:: State W
#    7250.000: /CONTROL_UNIT:: State W
#    7750.000: /CONTROL_UNIT:: State W
#    8250.000: /CONTROL_UNIT:: State W
#    8750.000: /CONTROL_UNIT:: State W
#    9000.000: /BUS_UNIT:: state J
#    9250.000: /MDR:: reading Data=0x00
#    9250.000: /CONTROL_UNIT:: State W
#    9500.000: /MDR:: reading Data=0x00
#    9500.000: /BUS_UNIT:: state J
#    9750.000: /MDR:: reading Data=0x00
#    9750.000: /CONTROL_UNIT:: State Y
#    10000.000: /BUS_UNIT:: state J
#    10250.000: /CONTROL_UNIT:: State Y
#    10500.000: /BUS_UNIT:: state F
#    10750.000: /CONTROL_UNIT:: State Y
#    11000.000: /BUS_UNIT:: State A
#    11000.000: /BUS_UNIT:: Pre-fetch empty...
#    11000.000: /BUS_UNIT:: Fill the prefetch
#    11250.000: /CONTROL_UNIT:: State V
#    11500.000: /BUS_UNIT:: State C
#    11750.000: /CONTROL_UNIT:: State V
#    12000.000: /BUS_UNIT:: State C
#    12250.000: /CONTROL_UNIT:: State V
#    12500.000: /BUS_UNIT:: state G
#    12750.000: /CONTROL_UNIT:: State V
#    13000.000: /BUS_UNIT:: state P
#    13250.000: /CONTROL_UNIT:: State V
#    13500.000: /BUS_UNIT:: state H
#    13750.000: /CONTROL_UNIT:: State V
```

```
#    14000.000: /BUS_UNIT:: State A
#    14000.000: /BUS_UNIT:: Q[2] = 0x01

#    14250.000: /CONTROL_UNIT:: State V
#    14500.000: /BUS_UNIT:: state J
#    14750.000: /CONTROL_UNIT:: State V
#    15000.000: /BUS_UNIT:: state J
#    15250.000: /CONTROL_UNIT:: State E
#    15255.000: REGA:: reading Data = 0x01
#    15500.000: REGA:: reading Data = 0x01
#    15500.000: /BUS_UNIT:: state J
#    15750.000: REGA:: reading Data = 0x01
#    15750.000: /CONTROL_UNIT:: State A
#    16000.000: /BUS_UNIT:: state J
#    16250.000: /CONTROL_UNIT:: State A
#    16500.000: /BUS_UNIT:: state F
#    16750.000: /CONTROL_UNIT:: State A
#    17000.000: /BUS_UNIT:: State A
#    17000.000: /BUS_UNIT:: Pre-fetch empty...
#    17000.000: /BUS_UNIT:: Fill the prefetch
#    17250.000: /CONTROL_UNIT:: State Q
#    17500.000: /BUS_UNIT:: State C
#    17750.000: /CONTROL_UNIT:: State Q
#    18000.000: /BUS_UNIT:: State C
#    18250.000: /CONTROL_UNIT:: State Q
#    18500.000: /BUS_UNIT:: state G
#    18750.000: /CONTROL_UNIT:: State Q
#    19000.000: /BUS_UNIT:: state P
#    19250.000: /CONTROL_UNIT:: State Q
#    19500.000: /BUS_UNIT:: state H
#    19750.000: /CONTROL_UNIT:: State Q
#    20000.000: /BUS_UNIT:: state J
#    20250.000: /CONTROL_UNIT:: State Q
#    20500.000: /BUS_UNIT:: state J
#    20750.000: /CONTROL_UNIT:: State R
#    21000.000: /INSTR_REG:: instr = 0xc8
#    21000.000: /INSTR_REG:: instr(2:0)=0x01
#    21000.000: /INSTR_REG:: decode type = 0x03
#    21000.000: /BUS_UNIT:: state J
#    21250.000: /INSTR_REG:: instr = 0xc8
#    21250.000: /INSTR_REG:: instr(2:0)=0x01
#    21250.000: /INSTR_REG:: decode type = 0x03
#    21250.000: /CONTROL_UNIT:: state U
#    21500.000: /BUS_UNIT:: state F
#    21750.000: /CONTROL_UNIT:: State G
#    21755.000: REGA:: Writing 0x01 on bus0
#    22000.000: REGA:: Writing 0x01 on bus0
```

```
#   22000.000: /BUS_UNIT:: State A
#   22000.000: /BUS_UNIT:: Pre-fetch empty...
#   22000.000: /BUS_UNIT:: Fill the prefetch
#   22005.000: REGA:: reading Data = 0x02
#   22250.000: REGA:: reading Data = 0x02
#   22250.000: /FLAGS_REG:: Just took i0 as 00
#   22250.000: /CONTROL_UNIT:: State A
#   22500.000: /BUS_UNIT:: State C
#   22750.000: /CONTROL_UNIT:: State Q
#   23000.000: /BUS_UNIT:: State C
#   23250.000: /CONTROL_UNIT:: State Q
#   23500.000: /BUS_UNIT:: state G
#   23750.000: /CONTROL_UNIT:: State Q
#   24000.000: /BUS_UNIT:: state P
#   24250.000: /CONTROL_UNIT:: State Q
#   24500.000: /BUS_UNIT:: state H
#   24750.000: /CONTROL_UNIT:: State Q
#   25000.000: /BUS_UNIT:: State A
#   25000.000: /BUS_UNIT:: Q[0] = 0x08

#   25250.000: /CONTROL_UNIT:: State Q
#   25500.000: /BUS_UNIT:: state J
#   25750.000: /CONTROL_UNIT:: State Q
#   26000.000: /BUS_UNIT:: state J
#   26250.000: /CONTROL_UNIT:: State R
#   26500.000: /INSTR_REG:: instr = 0x08
#   26500.000: /INSTR_REG:: instr(2:0)=0x01
#   26500.000: /INSTR_REG:: decode type = 0x00
#   26500.000: /BUS_UNIT:: state J
#   26750.000: /INSTR_REG:: instr = 0x08
#   26750.000: /INSTR_REG:: instr(2:0)=0x01
#   26750.000: /INSTR_REG:: decode type = 0x00
#   26750.000: /CONTROL_UNIT:: state U
#   27000.000: /BUS_UNIT:: state F
#   27250.000: /CONTROL_UNIT:: State W
#   27255.000: /MDR:: reading Data=0x08
#   27500.000: /MDR:: reading Data=0x08
#   27500.000: /BUS_UNIT:: State A
#   27500.000: /BUS_UNIT:: Pre-fetch empty...
#   27750.000: /CONTROL_UNIT:: State W
#   28250.000: /CONTROL_UNIT:: State W
#   28750.000: /CONTROL_UNIT:: State W
#   29250.000: /CONTROL_UNIT:: State W
#   29750.000: /CONTROL_UNIT:: State W
#   30000.000: /BUS_UNIT:: state H
#   30250.000: /CONTROL_UNIT:: State W
#   30500.000: /BUS_UNIT:: state J
```

```
#    30750.000: /MDR:: reading Data=0x10
#    30750.000: /CONTROL_UNIT:: State W
#    31000.000: /MDR:: reading Data=0x10
#    31000.000: /BUS_UNIT:: state J
#    31250.000: /MDR:: reading Data=0x10
#    31250.000: /CONTROL_UNIT:: State Y
#    31500.000: /BUS_UNIT:: state J
#    31750.000: /CONTROL_UNIT:: State Y
#    32000.000: /BUS_UNIT:: state F
#    32250.000: /CONTROL_UNIT:: State Y
#    32500.000: /BUS_UNIT:: State A
#    32500.000: /BUS_UNIT:: Pre-fetch empty...
#    32500.000: /BUS_UNIT:: Fill the prefetch
#    32750.000: /CONTROL_UNIT:: State V
#    33000.000: /BUS_UNIT:: State C
#    33250.000: /CONTROL_UNIT:: State V
#    33500.000: /BUS_UNIT:: State C
#    33750.000: /CONTROL_UNIT:: State V
#    34000.000: /BUS_UNIT:: state G
#    34250.000: /CONTROL_UNIT:: State V
#    34500.000: /BUS_UNIT:: state P
#    34750.000: /CONTROL_UNIT:: State V
#    35000.000: /BUS_UNIT:: state H
#    35250.000: /CONTROL_UNIT:: State V
#    35500.000: /BUS_UNIT:: State A
#    35500.000: /BUS_UNIT:: Q[2] = 0xf1

#    35750.000: /CONTROL_UNIT:: State V
#    36000.000: /BUS_UNIT:: state J
#    36250.000: /CONTROL_UNIT:: State V
#    36500.000: /BUS_UNIT:: state J
#    36750.000: /CONTROL_UNIT:: State E
#    36755.000: REGB:: Data = 0xf1
#    37000.000: REGB:: Data = 0xf1
#    37000.000: /BUS_UNIT:: state J
#    37250.000: REGB:: Data = 0xf1
#    37250.000: /CONTROL_UNIT:: State A
#    37500.000: /BUS_UNIT:: state J
#    37750.000: /CONTROL_UNIT:: State A
#    38000.000: /BUS_UNIT:: state F
#    38250.000: /CONTROL_UNIT:: State A
#    38500.000: /BUS_UNIT:: State A
#    38500.000: /BUS_UNIT:: Pre-fetch empty...
#    38500.000: /BUS_UNIT:: Fill the prefetch
#    38750.000: /CONTROL_UNIT:: State Q
#    39000.000: /BUS_UNIT:: State C
#    39250.000: /CONTROL_UNIT:: State Q
```

```
#    39500.000: /BUS_UNIT:: State C
#    39750.000: /CONTROL_UNIT:: State Q
#    40000.000: /BUS_UNIT:: state G
#    40250.000: /CONTROL_UNIT:: State Q
#    40500.000: /BUS_UNIT:: state P
#    40750.000: /CONTROL_UNIT:: State Q
#    41000.000: /BUS_UNIT:: state H
#    41250.000: /CONTROL_UNIT:: State Q
#    41500.000: /BUS_UNIT:: state J
#    41750.000: /CONTROL_UNIT:: State Q
#    42000.000: /BUS_UNIT:: state J
#    42250.000: /CONTROL_UNIT:: State R
#    42500.000: /INSTR_REG:: instr = 0xc9
#    42500.000: /INSTR_REG:: instr(2:0)=0x01
#    42500.000: /INSTR_REG:: decode type = 0x03
#    42500.000: /BUS_UNIT:: state J
#    42750.000: /INSTR_REG:: instr = 0xc9
#    42750.000: /INSTR_REG:: instr(2:0)=0x01
#    42750.000: /INSTR_REG:: decode type = 0x03
#    42750.000: /CONTROL_UNIT:: state U
#    43000.000: /BUS_UNIT:: state F
#    43250.000: /CONTROL_UNIT:: State G
#    43255.000: REGB:: Writing 0xf1 on bus0
#    43500.000: REGB:: Writing 0xf1 on bus0
#    43500.000: /BUS_UNIT:: State A
#    43500.000: /BUS_UNIT:: Pre-fetch empty...
#    43500.000: /BUS_UNIT:: Fill the prefetch
#    43505.000: REGB:: Data = 0xe2
#    43750.000: /FLAGS_REG:: Just took i0 as 09
#    43750.000: REGB:: Data = 0xe2
#    43750.000: /CONTROL_UNIT:: State A
#    44000.000: /BUS_UNIT:: State C
#    44250.000: /CONTROL_UNIT:: State Q
#    44500.000: /BUS_UNIT:: State C
#    44750.000: /CONTROL_UNIT:: State Q
#    45000.000: /BUS_UNIT:: state G
#    45250.000: /CONTROL_UNIT:: State Q
#    45500.000: /BUS_UNIT:: state P
#    45750.000: /CONTROL_UNIT:: State Q
#    46000.000: /BUS_UNIT:: state H
#    46250.000: /CONTROL_UNIT:: State Q
#    46500.000: /BUS_UNIT:: State A
#    46500.000: /BUS_UNIT:: Q[0] = 0x08

#    46750.000: /CONTROL_UNIT:: State Q
#    47000.000: /BUS_UNIT:: state J
#    47250.000: /CONTROL_UNIT:: State Q
```

```
#    47500.000: /BUS_UNIT:: state J
#    47750.000: /CONTROL_UNIT:: State R
#    48000.000: /INSTR_REG:: instr = 0x08
#    48000.000: /INSTR_REG:: instr(2:0)=0x01
#    48000.000: /INSTR_REG:: decode type = 0x00
#    48000.000: /BUS_UNIT:: state J
#    48250.000: /INSTR_REG:: instr = 0x08
#    48250.000: /INSTR_REG:: instr(2:0)=0x01
#    48250.000: /INSTR_REG:: decode type = 0x00
#    48250.000: /CONTROL_UNIT:: state U
#    48500.000: /BUS_UNIT:: state F
#    48750.000: /CONTROL_UNIT:: State W
#    48755.000: /MDR:: reading Data=0x08
#    49000.000: /MDR:: reading Data=0x08
#    49000.000: /BUS_UNIT:: State A
#    49000.000: /BUS_UNIT:: Pre-fetch empty...
#    49250.000: /CONTROL_UNIT:: State W
#    49500.000: /BUS_UNIT:: State C
#    49750.000: /CONTROL_UNIT:: State W
#    50000.000: /BUS_UNIT:: State C
#    50250.000: /CONTROL_UNIT:: State W
#    50500.000: /BUS_UNIT:: state G
#    50750.000: /CONTROL_UNIT:: State W
#    51000.000: /BUS_UNIT:: state P
#    51250.000: /CONTROL_UNIT:: State W
#    51500.000: /BUS_UNIT:: state H
#    51750.000: /CONTROL_UNIT:: State W
#    52000.000: /BUS_UNIT:: state J
#    52250.000: /MDR:: reading Data=0x20
#    52250.000: /CONTROL_UNIT:: State W
#    52500.000: /MDR:: reading Data=0x20
#    52500.000: /BUS_UNIT:: state J
#    52750.000: /MDR:: reading Data=0x20
#    52750.000: /CONTROL_UNIT:: State Y
#    53000.000: /BUS_UNIT:: state J
#    53250.000: /CONTROL_UNIT:: State Y
#    53500.000: /BUS_UNIT:: state F
#    53750.000: /CONTROL_UNIT:: State Y
#    54000.000: /BUS_UNIT:: State A
#    54000.000: /BUS_UNIT:: Pre-fetch empty...
#    54000.000: /BUS_UNIT:: Fill the prefetch
#    54250.000: /CONTROL_UNIT:: State V
#    54500.000: /BUS_UNIT:: State C
#    54750.000: /CONTROL_UNIT:: State V
#    55000.000: /BUS_UNIT:: State C
#    55250.000: /CONTROL_UNIT:: State V
#    55500.000: /BUS_UNIT:: state G
```

```
#    55750.000: /CONTROL_UNIT:: State V
#    56000.000: /BUS_UNIT:: state P
#    56250.000: /CONTROL_UNIT:: State V
#    56500.000: /BUS_UNIT:: state H
#    56750.000: /CONTROL_UNIT:: State V
#    57000.000: /BUS_UNIT:: State A
#    57000.000: /BUS_UNIT:: Q[2] = 0x91

#    57250.000: /CONTROL_UNIT:: State V
#    57500.000: /BUS_UNIT:: state J
#    57750.000: /CONTROL_UNIT:: State V
#    58000.000: /BUS_UNIT:: state J
#    58250.000: /CONTROL_UNIT:: State E
#    58255.000: REGC:: reading Data = 0x91
#    58500.000: REGC:: reading Data = 0x91
#    58500.000: /BUS_UNIT:: state J
#    58750.000: REGC:: reading Data = 0x91
#    58750.000: /CONTROL_UNIT:: State A
#    59000.000: /BUS_UNIT:: state J
#    59250.000: /CONTROL_UNIT:: State A
#    59500.000: /BUS_UNIT:: state F
#    59750.000: /CONTROL_UNIT:: State A
#    60000.000: /BUS_UNIT:: State A
#    60000.000: /BUS_UNIT:: Pre-fetch empty...
#    60000.000: /BUS_UNIT:: Fill the prefetch
#    60250.000: /CONTROL_UNIT:: State Q
#    60500.000: /BUS_UNIT:: State C
#    60750.000: /CONTROL_UNIT:: State Q
#    61000.000: /BUS_UNIT:: State C
#    61250.000: /CONTROL_UNIT:: State Q
#    61500.000: /BUS_UNIT:: state G
#    61750.000: /CONTROL_UNIT:: State Q
#    62000.000: /BUS_UNIT:: state P
#    62250.000: /CONTROL_UNIT:: State Q
#    62500.000: /BUS_UNIT:: state H
#    62750.000: /CONTROL_UNIT:: State Q
#    63000.000: /BUS_UNIT:: state J
#    63250.000: /CONTROL_UNIT:: State Q
#    63500.000: /BUS_UNIT:: state J
#    63750.000: /CONTROL_UNIT:: State R
#    64000.000: /INSTR_REG:: instr = 0xd2
#    64000.000: /INSTR_REG:: instr(2:0)=0x02
#    64000.000: /INSTR_REG:: decode type = 0x03
#    64000.000: /BUS_UNIT:: state J
#    64250.000: /INSTR_REG:: instr = 0xd2
#    64250.000: /INSTR_REG:: instr(2:0)=0x02
#    64250.000: /INSTR_REG:: decode type = 0x03
```

```
#   64250.000: /CONTROL_UNIT:: state U
#   64500.000: /BUS_UNIT:: state F
#   64750.000: /CONTROL_UNIT:: State G
#   64755.000: REGC:: Writing 0x91 on bus0
#   65000.000: REGC:: Writing 0x91 on bus0
#   65000.000: /BUS_UNIT:: State A
#   65000.000: /BUS_UNIT:: Pre-fetch empty...
#   65000.000: /BUS_UNIT:: Fill the prefetch
#   65005.000: REGC:: reading Data = 0xc8
#   65250.000: /FLAGS_REG:: Just took i0 as 09
#   65250.000: REGC:: reading Data = 0xc8
#   65250.000: /CONTROL_UNIT:: State A
#   65500.000: /BUS_UNIT:: State C
#   65750.000: /CONTROL_UNIT:: State Q
#   66000.000: /BUS_UNIT:: State C
#   66250.000: /CONTROL_UNIT:: State Q
#   66500.000: /BUS_UNIT:: state G
#   66750.000: /CONTROL_UNIT:: State Q
#   67000.000: /BUS_UNIT:: state P
#   67250.000: /CONTROL_UNIT:: State Q
#   67500.000: /BUS_UNIT:: state H
#   67750.000: /CONTROL_UNIT:: State Q
#   68000.000: /BUS_UNIT:: State A
#   68000.000: /BUS_UNIT:: Q[0] = 0xc2

#   68250.000: /CONTROL_UNIT:: State Q
#   68500.000: /BUS_UNIT:: state J
#   68750.000: /CONTROL_UNIT:: State Q
#   69000.000: /BUS_UNIT:: state J
#   69250.000: /CONTROL_UNIT:: State R
#   69500.000: /INSTR_REG:: instr = 0xc2
#   69500.000: /INSTR_REG:: instr(2:0)=0x00
#   69500.000: /INSTR_REG:: decode type = 0x03
#   69500.000: /BUS_UNIT:: state J
#   69750.000: /INSTR_REG:: instr = 0xc2
#   69750.000: /INSTR_REG:: instr(2:0)=0x00
#   69750.000: /INSTR_REG:: decode type = 0x03
#   69750.000: /CONTROL_UNIT:: state U
#   70000.000: /BUS_UNIT:: state F
#   70250.000: /CONTROL_UNIT:: State G
#   70255.000: REGC:: Writing 0xc8 on bus0
#   70500.000: REGC:: Writing 0xc8 on bus0
#   70500.000: /BUS_UNIT:: State A
#   70500.000: /BUS_UNIT:: Pre-fetch empty...
#   70500.000: /BUS_UNIT:: Fill the prefetch
#   70505.000: REGC:: reading Data = 0x64
#   70750.000: /FLAGS_REG:: Just took i0 as 00
```

```
#    70750.000: REGC:: reading Data = 0x64
#    70750.000: /CONTROL_UNIT:: State A
#    71000.000: /BUS_UNIT:: State C
#    71250.000: /CONTROL_UNIT:: State Q
#    71500.000: /BUS_UNIT:: State C
#    71750.000: /CONTROL_UNIT:: State Q
#    72000.000: /BUS_UNIT:: state G
#    72250.000: /CONTROL_UNIT:: State Q
#    72500.000: /BUS_UNIT:: state P
#    72750.000: /CONTROL_UNIT:: State Q
#    73000.000: /BUS_UNIT:: state H
#    73250.000: /CONTROL_UNIT:: State Q
#    73500.000: /BUS_UNIT:: State A
#    73500.000: /BUS_UNIT:: Q[1] = 0xb0

#    73750.000: /CONTROL_UNIT:: State Q
#    74000.000: /BUS_UNIT:: state J
#    74250.000: /CONTROL_UNIT:: State Q
#    74500.000: /BUS_UNIT:: state J
#    74750.000: /CONTROL_UNIT:: State R
#    75000.000: /INSTR_REG:: instr = 0xb0
#    75000.000: /INSTR_REG:: instr(2:0)=0x06
#    75000.000: /INSTR_REG:: decode type = 0x02
#    75000.000: /BUS_UNIT:: state J
#    75250.000: /INSTR_REG:: instr = 0xb0
#    75250.000: /INSTR_REG:: instr(2:0)=0x06
#    75250.000: /INSTR_REG:: decode type = 0x02
#    75250.000: /CONTROL_UNIT:: state U
#    75500.000: /BUS_UNIT:: state F
#    75750.000: /CONTROL_UNIT:: State M
#    76000.000: /FLAGS_REG:: Data now = 0x01
#    76000.000: /BUS_UNIT:: State A
#    76000.000: /BUS_UNIT:: Pre-fetch empty...
#    76000.000: /BUS_UNIT:: Fill the prefetch
#    76250.000: /FLAGS_REG:: Data now = 0x01
#    76250.000: /CONTROL_UNIT:: State A
#    76500.000: /BUS_UNIT:: State C
#    76750.000: /CONTROL_UNIT:: State Q
#    77000.000: /BUS_UNIT:: State C
#    77250.000: /CONTROL_UNIT:: State Q
#    77500.000: /BUS_UNIT:: state G
#    77750.000: /CONTROL_UNIT:: State Q
#    78000.000: /BUS_UNIT:: state P
#    78250.000: /CONTROL_UNIT:: State Q
#    78500.000: /BUS_UNIT:: state H
#    78750.000: /CONTROL_UNIT:: State Q
#    79000.000: /BUS_UNIT:: State A
```

```
#   79000.000: /BUS_UNIT:: Q[2] = 0xe0

#   79250.000: /CONTROL_UNIT:: State Q
#   79500.000: /BUS_UNIT:: state J
#   79750.000: /CONTROL_UNIT:: State Q
#   80000.000: /BUS_UNIT:: state J
#   80250.000: /CONTROL_UNIT:: State R
#   80500.000: /INSTR_REG:: instr = 0xe0
#   80500.000: /INSTR_REG:: instr(2:0)=0x04
#   80500.000: /INSTR_REG:: decode type = 0x03
#   80500.000: /BUS_UNIT:: state J
#   80750.000: /INSTR_REG:: instr = 0xe0
#   80750.000: /INSTR_REG:: instr(2:0)=0x04
#   80750.000: /INSTR_REG:: decode type = 0x03
#   80750.000: /CONTROL_UNIT:: state U
#   81000.000: /BUS_UNIT:: state F
#   81250.000: /CONTROL_UNIT:: State G
#   81255.000: REGC:: Writing 0x64 on bus0
#   81500.000: REGC:: Writing 0x64 on bus0
#   81500.000: /BUS_UNIT:: State A
#   81500.000: /BUS_UNIT:: Pre-fetch empty...
#   81500.000: /BUS_UNIT:: Fill the prefetch
#   81750.000: /FLAGS_REG:: Just took i0 as 00
#   81750.000: REGC:: reading Data = 0x32
#   81750.000: /CONTROL_UNIT:: State A
#   82000.000: /BUS_UNIT:: State C
#   82250.000: /CONTROL_UNIT:: State Q
#   82500.000: /BUS_UNIT:: State C
#   82750.000: /CONTROL_UNIT:: State Q
#   83000.000: /BUS_UNIT:: state G
#   83250.000: /CONTROL_UNIT:: State Q
#   83500.000: /BUS_UNIT:: state P
#   83750.000: /CONTROL_UNIT:: State Q
#   84000.000: /BUS_UNIT:: state H
#   84250.000: /CONTROL_UNIT:: State Q
#   84500.000: /BUS_UNIT:: State A
#   84500.000: /BUS_UNIT:: Q[3] = 0xb0

#   84750.000: /CONTROL_UNIT:: State Q
#   85000.000: /BUS_UNIT:: state J
#   85250.000: /CONTROL_UNIT:: State Q
#   85500.000: /BUS_UNIT:: state J
#   85750.000: /CONTROL_UNIT:: State R
#   86000.000: /INSTR_REG:: instr = 0xb0
#   86000.000: /INSTR_REG:: instr(2:0)=0x06
#   86000.000: /INSTR_REG:: decode type = 0x02
#   86000.000: /BUS_UNIT:: state J
```

```
#    86250.000: /INSTR_REG:: instr = 0xb0
#    86250.000: /INSTR_REG:: instr(2:0)=0x06
#    86250.000: /INSTR_REG:: decode type = 0x02
#    86250.000: /CONTROL_UNIT:: state U
#    86500.000: /BUS_UNIT:: state F
#    86750.000: /CONTROL_UNIT:: State M
#    87000.000: /FLAGS_REG:: Data now = 0x01
#    87000.000: /BUS_UNIT:: State A
#    87000.000: /BUS_UNIT:: Pre-fetch empty...
#    87000.000: /BUS_UNIT:: Fill the prefetch
#    87250.000: /FLAGS_REG:: Data now = 0x01
#    87250.000: /CONTROL_UNIT:: State A
#    87500.000: /BUS_UNIT:: State C
#    87750.000: /CONTROL_UNIT:: State Q
#    88000.000: /BUS_UNIT:: State C
#    88250.000: /CONTROL_UNIT:: State Q
#    88500.000: /BUS_UNIT:: state G
#    88750.000: /CONTROL_UNIT:: State Q
#    89000.000: /BUS_UNIT:: state P
#    89250.000: /CONTROL_UNIT:: State Q
#    89500.000: /BUS_UNIT:: state H
#    89750.000: /CONTROL_UNIT:: State Q
#    90000.000: /BUS_UNIT:: State A
#    90000.000: /BUS_UNIT:: Q[0] = 0xe9

#    90250.000: /CONTROL_UNIT:: State Q
#    90500.000: /BUS_UNIT:: state J
#    90750.000: /CONTROL_UNIT:: State Q
#    91000.000: /BUS_UNIT:: state J
#    91250.000: /CONTROL_UNIT:: State R
#    91500.000: /INSTR_REG:: instr = 0xe9
#    91500.000: /INSTR_REG:: instr(2:0)=0x05
#    91500.000: /INSTR_REG:: decode type = 0x03
#    91500.000: /BUS_UNIT:: state J
#    91750.000: /INSTR_REG:: instr = 0xe9
#    91750.000: /INSTR_REG:: instr(2:0)=0x05
#    91750.000: /INSTR_REG:: decode type = 0x03
#    91750.000: /CONTROL_UNIT:: state U
#    92000.000: /BUS_UNIT:: state F
#    92250.000: /CONTROL_UNIT:: State G
#    92255.000: REGC:: Writing 0x32 on bus0
#    92500.000: REGC:: Writing 0x32 on bus0
#    92500.000: /BUS_UNIT:: State A
#    92500.000: /BUS_UNIT:: Pre-fetch empty...
#    92500.000: /BUS_UNIT:: Fill the prefetch
#    92750.000: /FLAGS_REG:: Just took i0 as 00
#    92750.000: REGC:: reading Data = 0x64
```

```
#    92750.000: /CONTROL_UNIT:: State A
#    93000.000: /BUS_UNIT:: State C
#    93250.000: /CONTROL_UNIT:: State Q
#    93500.000: /BUS_UNIT:: State C
#    93750.000: /CONTROL_UNIT:: State Q
#    94000.000: /BUS_UNIT:: state G
#    94250.000: /CONTROL_UNIT:: State Q
#    94500.000: /BUS_UNIT:: state P
#    94750.000: /CONTROL_UNIT:: State Q
#    95000.000: /BUS_UNIT:: state H
#    95250.000: /CONTROL_UNIT:: State Q
#    95500.000: /BUS_UNIT:: State A
#    95500.000: /BUS_UNIT:: Q[1] = 0xb0

#    95750.000: /CONTROL_UNIT:: State Q
#    96000.000: /BUS_UNIT:: state J
#    96250.000: /CONTROL_UNIT:: State Q
#    96500.000: /BUS_UNIT:: state J
#    96750.000: /CONTROL_UNIT:: State R
#    97000.000: /INSTR_REG:: instr = 0xb0
#    97000.000: /INSTR_REG:: instr(2:0)=0x06
#    97000.000: /INSTR_REG:: decode type = 0x02
#    97000.000: /BUS_UNIT:: state J
#    97250.000: /INSTR_REG:: instr = 0xb0
#    97250.000: /INSTR_REG:: instr(2:0)=0x06
#    97250.000: /INSTR_REG:: decode type = 0x02
#    97250.000: /CONTROL_UNIT:: state U
#    97500.000: /BUS_UNIT:: state F
#    97750.000: /CONTROL_UNIT:: State M
#    98000.000: /FLAGS_REG:: Data now = 0x01
#    98000.000: /BUS_UNIT:: State A
#    98000.000: /BUS_UNIT:: Pre-fetch empty...
#    98000.000: /BUS_UNIT:: Fill the prefetch
#    98250.000: /FLAGS_REG:: Data now = 0x01
#    98250.000: /CONTROL_UNIT:: State A
#    98500.000: /BUS_UNIT:: State C
#    98750.000: /CONTROL_UNIT:: State Q
#    99000.000: /BUS_UNIT:: State C
#    99250.000: /CONTROL_UNIT:: State Q
#    99500.000: /BUS_UNIT:: state G
#    99750.000: /CONTROL_UNIT:: State Q
#    100000.000: /BUS_UNIT:: state P
#    100250.000: /CONTROL_UNIT:: State Q
#    100500.000: /BUS_UNIT:: state H
#    100750.000: /CONTROL_UNIT:: State Q
#    101000.000: /BUS_UNIT:: State A
#    101000.000: /BUS_UNIT:: Q[2] = 0xa8
```

```
#    101250.000: /CONTROL_UNIT:: State Q
#    101500.000: /BUS_UNIT:: state J
#    101750.000: /CONTROL_UNIT:: State Q
#    102000.000: /BUS_UNIT:: state J
#    102250.000: /CONTROL_UNIT:: State R
#    102500.000: /INSTR_REG:: instr = 0xa8
#    102500.000: /INSTR_REG:: instr(2:0)=0x05
#    102500.000: /INSTR_REG:: decode type = 0x02
#    102500.000: /BUS_UNIT:: state J
#    102750.000: /INSTR_REG:: instr = 0xa8
#    102750.000: /INSTR_REG:: instr(2:0)=0x05
#    102750.000: /INSTR_REG:: decode type = 0x02
#    102750.000: /CONTROL_UNIT:: state U
#    103000.000: /BUS_UNIT:: state F
#    103250.000: /CONTROL_UNIT:: State N
#    103500.000: /FLAGS_REG:: Data now = 0x00
```

```
#
# This program is used to test the unconditional
# jump instruction.  The choice of the mult
# instruction is not important.
#
00/01;
01/08;  #lda #$02
02/00;
03/02;
04/08;  #ldb #$03
05/10;
06/03;
07/b8;  #jump $0a
08/0a;
09/a0;  #halt
0a/98;  #mult  c=#$00, d=#$06
0b/00;
0c/00;
0d/00;
0e/00;



#   500.000: /BUS_UNIT:: Reset Activated
#   1000.000: /BUS_UNIT:: State C
#   2000.000: /BUS_UNIT:: State C
#   3000.000: /BUS_UNIT:: state G
#   4000.000: /BUS_UNIT:: state P
#   5000.000: /BUS_UNIT:: state I
#   6000.000: /BUS_UNIT:: state O
#   7000.000: /BUS_UNIT:: state L
#   8000.000: /BUS_UNIT:: state D
#   9000.000: /BUS_UNIT:: state N
#   10000.000: /BUS_UNIT:: state N
#   11000.000: /INSTR_REG:: instr = 0x08
#   11000.000: /INSTR_REG:: instr(2:0)=0x01
#   11000.000: /INSTR_REG:: decode type = 0x00
#   11000.000: /BUS_UNIT:: state N
#   11500.000: /INSTR_REG:: instr = 0x08
#   11500.000: /INSTR_REG:: instr(2:0)=0x01
#   11500.000: /INSTR_REG:: decode type = 0x00
#   12000.000: /BUS_UNIT:: State A
#   12000.000: /BUS_UNIT:: Pre-fetch empty...
#   12000.000: /BUS_UNIT:: Fill the prefetch
#   13000.000: /BUS_UNIT:: State C
#   14000.000: /BUS_UNIT:: State C
```

```
#   15000.000: /BUS_UNIT:: state G
#   16000.000: /BUS_UNIT:: state P
#   17000.000: /BUS_UNIT:: state H
#   18000.000: /BUS_UNIT:: state J
#   18500.000: /MDR:: reading Data=0x00
#   19000.000: /MDR:: reading Data=0x00
#   19000.000: /BUS_UNIT:: state J
#   19500.000: /MDR:: reading Data=0x00
#   20000.000: /BUS_UNIT:: state J
#   21000.000: /BUS_UNIT:: state F
#   22000.000: /BUS_UNIT:: State A
#   22000.000: /BUS_UNIT:: Pre-fetch empty...
#   22000.000: /BUS_UNIT:: Fill the prefetch
#   23000.000: /BUS_UNIT:: State C
#   24000.000: /BUS_UNIT:: State C
#   25000.000: /BUS_UNIT:: state G
#   26000.000: /BUS_UNIT:: state P
#   27000.000: /BUS_UNIT:: state H
#   28000.000: /BUS_UNIT:: State A
#   28000.000: /BUS_UNIT:: Q[2] = 0x02

#   29000.000: /BUS_UNIT:: state J
#   30000.000: /BUS_UNIT:: state J
#   30505.000: REGA:: reading Data = 0x02
#   31000.000: REGA:: reading Data = 0x02
#   31000.000: /BUS_UNIT:: state J
#   31500.000: REGA:: reading Data = 0x02
#   32000.000: /BUS_UNIT:: state J
#   33000.000: /BUS_UNIT:: state F
#   34000.000: /BUS_UNIT:: State A
#   34000.000: /BUS_UNIT:: Pre-fetch empty...
#   34000.000: /BUS_UNIT:: Fill the prefetch
#   35000.000: /BUS_UNIT:: State C
#   36000.000: /BUS_UNIT:: State C
#   37000.000: /BUS_UNIT:: state G
#   38000.000: /BUS_UNIT:: state P
#   39000.000: /BUS_UNIT:: state H
#   40000.000: /BUS_UNIT:: state J
#   41000.000: /BUS_UNIT:: state J
#   42000.000: /INSTR_REG:: instr = 0x08
#   42000.000: /INSTR_REG:: instr(2:0)=0x01
#   42000.000: /INSTR_REG:: decode type = 0x00
#   42000.000: /BUS_UNIT:: state J
#   42500.000: /INSTR_REG:: instr = 0x08
#   42500.000: /INSTR_REG:: instr(2:0)=0x01
#   42500.000: /INSTR_REG:: decode type = 0x00
#   43000.000: /BUS_UNIT:: state F
```

```
#   43505.000: /MDR:: reading Data=0x08
#   44000.000: /MDR:: reading Data=0x08
#   44000.000: /BUS_UNIT:: State A
#   44000.000: /BUS_UNIT:: Pre-fetch empty...
#   45000.000: /BUS_UNIT:: State C
#   46000.000: /BUS_UNIT:: State C
#   47000.000: /BUS_UNIT:: state G
#   48000.000: /BUS_UNIT:: state P
#   49000.000: /BUS_UNIT:: state H
#   50000.000: /BUS_UNIT:: state J
#   50500.000: /MDR:: reading Data=0x10
#   51000.000: /MDR:: reading Data=0x10
#   51000.000: /BUS_UNIT:: state J
#   51500.000: /MDR:: reading Data=0x10
#   52000.000: /BUS_UNIT:: state J
#   53000.000: /BUS_UNIT:: state F
#   54000.000: /BUS_UNIT:: State A
#   54000.000: /BUS_UNIT:: Pre-fetch empty...
#   54000.000: /BUS_UNIT:: Fill the prefetch
#   55000.000: /BUS_UNIT:: State C
#   56000.000: /BUS_UNIT:: State C
#   57000.000: /BUS_UNIT:: state G
#   58000.000: /BUS_UNIT:: state P
#   59000.000: /BUS_UNIT:: state H
#   60000.000: /BUS_UNIT:: State A
#   60000.000: /BUS_UNIT:: Q[1] = 0x03

#   61000.000: /BUS_UNIT:: state J
#   62000.000: /BUS_UNIT:: state J
#   62505.000: REGB:: Data = 0x03
#   63000.000: REGB:: Data = 0x03
#   63000.000: /BUS_UNIT:: state J
#   63500.000: REGB:: Data = 0x03
#   64000.000: /BUS_UNIT:: state J
#   65000.000: /BUS_UNIT:: state F
#   66000.000: /BUS_UNIT:: State A
#   66000.000: /BUS_UNIT:: Pre-fetch empty...
#   66000.000: /BUS_UNIT:: Fill the prefetch
#   67000.000: /BUS_UNIT:: State C
#   68000.000: /BUS_UNIT:: State C
#   69000.000: /BUS_UNIT:: state G
#   70000.000: /BUS_UNIT:: state P
#   71000.000: /BUS_UNIT:: state H
#   72000.000: /BUS_UNIT:: state J
#   73000.000: /BUS_UNIT:: state J
#   74000.000: /INSTR_REG:: instr = 0xb8
#   74000.000: /INSTR_REG:: instr(2:0)=0x07
```

```
#   74000.000: /INSTR_REG:: decode type = 0x02
#   74000.000: /BUS_UNIT:: state J
#   74500.000: /INSTR_REG:: instr = 0xb8
#   74500.000: /INSTR_REG:: instr(2:0)=0x07
#   74500.000: /INSTR_REG:: decode type = 0x02
#   75000.000: /BUS_UNIT:: state F
#   75505.000: /MDR:: reading Data=0xb8
#   76000.000: /MDR:: reading Data=0xb8
#   76000.000: /BUS_UNIT:: State A
#   76000.000: /BUS_UNIT:: Pre-fetch empty...
#   77000.000: /BUS_UNIT:: State C
#   78000.000: /BUS_UNIT:: State C
#   79000.000: /BUS_UNIT:: state G
#   80000.000: /BUS_UNIT:: state P
#   81000.000: /BUS_UNIT:: state H
#   82000.000: /BUS_UNIT:: state J
#   82500.000: /MDR:: reading Data=0x0a
#   83000.000: /MDR:: reading Data=0x0a
#   83000.000: /BUS_UNIT:: state J
#   83500.000: /MDR:: reading Data=0x0a
#   84000.000: /BUS_UNIT:: state J
#   85000.000: /BUS_UNIT:: state F
#   86000.000: /BUS_UNIT:: State A
#   86000.000: /BUS_UNIT:: Pre-fetch empty...
#   86000.000: /BUS_UNIT:: Fill the prefetch
#   86505.000: /MDR:: writing Data=0x0a
#   87000.000: /MDR:: writing Data=0x0a
#   87000.000: /BUS_UNIT:: State C
#   87500.000: /MDR:: writing Data=0x0a
#   88000.000: /MDR:: writing Data=0x0a
#   88000.000: /BUS_UNIT:: State C
#   88500.000: /MDR:: writing Data=0x0a
#   89000.000: /MDR:: writing Data=0x0a
#   89000.000: /BUS_UNIT:: state G
#   89500.000: /MDR:: writing Data=0x0a
#   90000.000: /MDR:: writing Data=0x0a
#   90000.000: /BUS_UNIT:: state P
#   90500.000: /MDR:: writing Data=0x0a
#   91000.000: /MDR:: writing Data=0x0a
#   91000.000: /BUS_UNIT:: state H
#   91500.000: /MDR:: writing Data=0x0a
#   92000.000: /MDR:: writing Data=0x0a
#   92000.000: /BUS_UNIT:: State A
#   92000.000: /BUS_UNIT:: Q[0] = 0xa0

#   92500.000: /MDR:: writing Data=0x0a
#   93000.000: /MDR:: writing Data=0x0a
```

```
#   93000.000: /BUS_UNIT:: state B
#   93000.000: /BUS_UNIT:: New addr = a

#   93500.000: /MDR:: writing Data=0x0a
#   94000.000: /MDR:: writing Data=0x0a
#   94000.000: /BUS_UNIT:: state B
#   94000.000: /BUS_UNIT:: New addr = a

#   94500.000: /MDR:: writing Data=0x0a
#   95000.000: /BUS_UNIT:: state B
#   96000.000: /BUS_UNIT:: State A
#   96000.000: /BUS_UNIT:: Pre-fetch empty...
#   96000.000: /BUS_UNIT:: Fill the prefetch
#   97000.000: /BUS_UNIT:: State C
#   98000.000: /BUS_UNIT:: State C
#   99000.000: /BUS_UNIT:: state G
#   100000.000: /BUS_UNIT:: state P
#   101000.000: /BUS_UNIT:: state H
#   102000.000: /BUS_UNIT:: State A
#   102000.000: /BUS_UNIT:: Q[0] = 0x98

#   103000.000: /BUS_UNIT:: state J
#   104000.000: /BUS_UNIT:: state J
#   105000.000: /INSTR_REG:: instr = 0x98
#   105000.000: /INSTR_REG:: instr(2:0)=0x03
#   105000.000: /INSTR_REG:: decode type = 0x02
#   105000.000: /BUS_UNIT:: state J
#   105500.000: /INSTR_REG:: instr = 0x98
#   105500.000: /INSTR_REG:: instr(2:0)=0x03
#   105500.000: /INSTR_REG:: decode type = 0x02
#   106000.000: /BUS_UNIT:: state F
#   106505.000: REGA:: Writing 0x02 on bus0
#   106505.000: REGB:: Writing 0x03 on bus1
#   107000.000: REGA:: Writing 0x02 on bus0
#   107000.000: REGB:: Writing 0x03 on bus1
#   107000.000: REGD:: Data = 0x06
```

```
#
# This program tests whether or not the conditional
# jumps work correctly.  None of the jumps should
# occur since the Flags Register is set to $00.
#
00/01;
01/08;  #ld flags #$00
02/40;
03/00;
04/b9;  #jmpz  #$aa (jump if z=1)
05/aa;
06/ba;  #jmpo  #$ab (jump if over/under)
07/ab;
08/bb;  #jmpc  #$ac (jump if carry)
09/ac;
0a/bc;  #jmpn  #$ad (jump if neg)
0b/ad;
0c/a0;  #halt
0d/00;
0e/00;
0f/00;
10/00;
11/00;
aa/a0;
ab/a0;
ac/a0;
ad/a0;


#    500.000: /BUS_UNIT:: Reset Activated
#    500.000: /BUS_UNIT:: State C
#   1000.000: /BUS_UNIT:: State C
#   1500.000: /BUS_UNIT:: state G
#   2000.000: /BUS_UNIT:: state P
#   2500.000: /BUS_UNIT:: state I
#   3000.000: /BUS_UNIT:: state O
#   3500.000: /BUS_UNIT:: state L
#   4000.000: /BUS_UNIT:: state D
#   4500.000: /BUS_UNIT:: state N
#   5000.000: /BUS_UNIT:: state N
#   5500.000: /INSTR_REG:: instr = 0x08
#   5500.000: /INSTR_REG:: instr(2:0)=0x01
#   5500.000: /INSTR_REG:: decode type = 0x00
#   5500.000: /BUS_UNIT:: state N
#   5750.000: /INSTR_REG:: instr = 0x08
#   5750.000: /INSTR_REG:: instr(2:0)=0x01
#   5750.000: /INSTR_REG:: decode type = 0x00
```

```
#    6000.000: /BUS_UNIT:: State A
#    6000.000: /BUS_UNIT:: Pre-fetch empty...
#    6000.000: /BUS_UNIT:: Fill the prefetch
#    6500.000: /BUS_UNIT:: State C
#    7000.000: /BUS_UNIT:: State C
#    7500.000: /BUS_UNIT:: state G
#    8000.000: /BUS_UNIT:: state P
#    8500.000: /BUS_UNIT:: state H
#    9000.000: /BUS_UNIT:: state J
#    9250.000: /MDR:: reading Data=0x40
#    9500.000: /MDR:: reading Data=0x40
#    9500.000: /BUS_UNIT:: state J
#    9750.000: /MDR:: reading Data=0x40
#    10000.000: /BUS_UNIT:: state J
#    10500.000: /BUS_UNIT:: state F
#    11000.000: /BUS_UNIT:: State A
#    11000.000: /BUS_UNIT:: Pre-fetch empty...
#    11000.000: /BUS_UNIT:: Fill the prefetch
#    11500.000: /BUS_UNIT:: State C
#    12000.000: /BUS_UNIT:: State C
#    12500.000: /BUS_UNIT:: state G
#    13000.000: /BUS_UNIT:: state P
#    13500.000: /BUS_UNIT:: state H
#    14000.000: /BUS_UNIT:: State A
#    14000.000: /BUS_UNIT:: Q[2] = 0x00

#    14500.000: /BUS_UNIT:: state J
#    15000.000: /BUS_UNIT:: state J
#    15500.000: /FLAGS_REG:: Just took 1i as 00
#    15500.000: /BUS_UNIT:: state J
#    15750.000: /FLAGS_REG:: Just took 1i as 00
#    16000.000: /BUS_UNIT:: state J
#    16500.000: /BUS_UNIT:: state F
#    17000.000: /BUS_UNIT:: State A
#    17000.000: /BUS_UNIT:: Pre-fetch empty...
#    17000.000: /BUS_UNIT:: Fill the prefetch
#    17500.000: /BUS_UNIT:: State C
#    18000.000: /BUS_UNIT:: State C
#    18500.000: /BUS_UNIT:: state G
#    19000.000: /BUS_UNIT:: state P
#    19500.000: /BUS_UNIT:: state H
#    20000.000: /BUS_UNIT:: state J
#    20500.000: /BUS_UNIT:: state J
#    21000.000: /INSTR_REG:: instr = 0xb9
#    21000.000: /INSTR_REG:: instr(2:0)=0x07
#    21000.000: /INSTR_REG:: decode type = 0x02
#    21000.000: /BUS_UNIT:: state J
```

```
#   21250.000: /INSTR_REG:: instr = 0xb9
#   21250.000: /INSTR_REG:: instr(2:0)=0x07
#   21250.000: /INSTR_REG:: decode type = 0x02
#   21500.000: /BUS_UNIT:: state F
#   21755.000: /MDR:: reading Data=0xb9
#   22000.000: /MDR:: reading Data=0xb9
#   22000.000: /BUS_UNIT:: State A
#   22000.000: /BUS_UNIT:: Pre-fetch empty...
#   22500.000: /BUS_UNIT:: State C
#   23000.000: /BUS_UNIT:: State C
#   23500.000: /BUS_UNIT:: state G
#   24000.000: /BUS_UNIT:: state P
#   24500.000: /BUS_UNIT:: state H
#   25000.000: /BUS_UNIT:: state J
#   25250.000: /MDR:: reading Data=0xaa
#   25500.000: /MDR:: reading Data=0xaa
#   25500.000: /BUS_UNIT:: state J
#   25750.000: /MDR:: reading Data=0xaa
#   26000.000: /BUS_UNIT:: state J
#   26500.000: /BUS_UNIT:: state F
#   27000.000: /BUS_UNIT:: State A
#   27000.000: /BUS_UNIT:: Pre-fetch empty...
#   27000.000: /BUS_UNIT:: Fi11 the prefetch
#   27500.000: /BUS_UNIT:: State C
#   28000.000: /BUS_UNIT:: State C
#   28500.000: /BUS_UNIT:: state G
#   29000.000: /BUS_UNIT:: state P
#   29500.000: /BUS_UNIT:: state H
#   30000.000: /BUS_UNIT:: State A
#   30000.000: /BUS_UNIT:: Q[1] = 0xba

#   30500.000: /BUS_UNIT:: state J
#   31000.000: /BUS_UNIT:: state J
#   31500.000: /INSTR_REG:: instr = 0xba
#   31500.000: /INSTR_REG:: instr(2:0)=0x07
#   31500.000: /INSTR_REG:: decode type = 0x02
#   31500.000: /BUS_UNIT:: state J
#   31750.000: /INSTR_REG:: instr = 0xba
#   31750.000: /INSTR_REG:: instr(2:0)=0x07
#   31750.000: /INSTR_REG:: decode type = 0x02
#   32000.000: /BUS_UNIT:: state F
#   32255.000: /MDR:: reading Data=0xba
#   32500.000: /MDR:: reading Data=0xba
#   32500.000: /BUS_UNIT:: State A
#   32500.000: /BUS_UNIT:: Pre-fetch empty...
#   33000.000: /BUS_UNIT:: State C
#   33500.000: /BUS_UNIT:: State C
```

```
#    34000.000: /BUS_UNIT:: state G
#    34500.000: /BUS_UNIT:: state P
#    35000.000: /BUS_UNIT:: state H
#    35500.000: /BUS_UNIT:: state J
#    35750.000: /MDR:: reading Data=0xab
#    36000.000: /MDR:: reading Data=0xab
#    36000.000: /BUS_UNIT:: state J
#    36250.000: /MDR:: reading Data=0xab
#    36500.000: /BUS_UNIT:: state J
#    37000.000: /BUS_UNIT:: state F
#    37500.000: /BUS_UNIT:: State A
#    37500.000: /BUS_UNIT:: Pre-fetch empty...
#    37500.000: /BUS_UNIT:: Fill the prefetch
#    38000.000: /BUS_UNIT:: State C
#    38500.000: /BUS_UNIT:: State C
#    39000.000: /BUS_UNIT:: state G
#    39500.000: /BUS_UNIT:: state P
#    40000.000: /BUS_UNIT:: state H
#    40500.000: /BUS_UNIT:: State A
#    40500.000: /BUS_UNIT:: Q[3] = 0xbb

#    41000.000: /BUS_UNIT:: state J
#    41500.000: /BUS_UNIT:: state J
#    42000.000: /INSTR_REG:: instr = 0xbb
#    42000.000: /INSTR_REG:: instr(2:0)=0x07
#    42000.000: /INSTR_REG:: decode type = 0x02
#    42000.000: /BUS_UNIT:: state J
#    42250.000: /INSTR_REG:: instr = 0xbb
#    42250.000: /INSTR_REG:: instr(2:0)=0x07
#    42250.000: /INSTR_REG:: decode type = 0x02
#    42500.000: /BUS_UNIT:: state F
#    42755.000: /MDR:: reading Data=0xbb
#    43000.000: /MDR:: reading Data=0xbb
#    43000.000: /BUS_UNIT:: State A
#    43000.000: /BUS_UNIT:: Pre-fetch empty...
#    43500.000: /BUS_UNIT:: State C
#    44000.000: /BUS_UNIT:: State C
#    44500.000: /BUS_UNIT:: state G
#    45000.000: /BUS_UNIT:: state P
#    45500.000: /BUS_UNIT:: state H
#    46000.000: /BUS_UNIT:: state J
#    46250.000: /MDR:: reading Data=0xac
#    46500.000: /MDR:: reading Data=0xac
#    46500.000: /BUS_UNIT:: state J
#    46750.000: /MDR:: reading Data=0xac
#    47000.000: /BUS_UNIT:: state J
#    47500.000: /BUS_UNIT:: state F
```

```
#    48000.000: /BUS_UNIT:: State A
#    48000.000: /BUS_UNIT:: Pre-fetch empty...
#    48000.000: /BUS_UNIT:: Fill the prefetch
#    48500.000: /BUS_UNIT:: State C
#    49000.000: /BUS_UNIT:: State C
#    49500.000: /BUS_UNIT:: state G
#    50000.000: /BUS_UNIT:: state P
#    50500.000: /BUS_UNIT:: state H
#    51000.000: /BUS_UNIT:: State A
#    51000.000: /BUS_UNIT:: Q[1] = 0xbc

#    51500.000: /BUS_UNIT:: state J
#    52000.000: /BUS_UNIT:: state J
#    52500.000: /INSTR_REG:: instr = 0xbc
#    52500.000: /INSTR_REG:: instr(2:0)=0x07
#    52500.000: /INSTR_REG:: decode type = 0x02
#    52500.000: /BUS_UNIT:: state J
#    52750.000: /INSTR_REG:: instr = 0xbc
#    52750.000: /INSTR_REG:: instr(2:0)=0x07
#    52750.000: /INSTR_REG:: decode type = 0x02
#    53000.000: /BUS_UNIT:: state F
#    53255.000: /MDR:: reading Data=0xbc
#    53500.000: /MDR:: reading Data=0xbc
#    53500.000: /BUS_UNIT:: State A
#    53500.000: /BUS_UNIT:: Pre-fetch empty...
#    54000.000: /BUS_UNIT:: State C
#    54500.000: /BUS_UNIT:: State C
#    55000.000: /BUS_UNIT:: state G
#    55500.000: /BUS_UNIT:: state P
#    56000.000: /BUS_UNIT:: state H
#    56500.000: /BUS_UNIT:: state J
#    56750.000: /MDR:: reading Data=0xad
#    57000.000: /MDR:: reading Data=0xad
#    57000.000: /BUS_UNIT:: state J
#    57250.000: /MDR:: reading Data=0xad
#    57500.000: /BUS_UNIT:: state J
#    58000.000: /BUS_UNIT:: state F
#    58500.000: /BUS_UNIT:: State A
#    58500.000: /BUS_UNIT:: Pre-fetch empty...
#    58500.000: /BUS_UNIT:: Fill the prefetch
#    59000.000: /BUS_UNIT:: State C
#    59500.000: /BUS_UNIT:: State C
#    60000.000: /BUS_UNIT:: state G
#    60500.000: /BUS_UNIT:: state P
#    61000.000: /BUS_UNIT:: state H
#    61500.000: /BUS_UNIT:: State A
#    61500.000: /BUS_UNIT:: Q[3] = 0xa0
```

.

```
#    62000.000: /BUS_UNIT:: state J
#    62500.000: /BUS_UNIT:: state J
#    63000.000: /INSTR_REG:: instr = 0xa0
#    63000.000: /INSTR_REG:: instr(2:0)=0x04
#    63000.000: /INSTR_REG:: decode type = 0x02
#    63000.000: /BUS_UNIT:: state J
#    63250.000: /INSTR_REG:: instr = 0xa0
#    63250.000: /INSTR_REG:: instr(2:0)=0x04
#    63250.000: /INSTR_REG:: decode type = 0x02
#    63500.000: /BUS_UNIT:: state F
#    64000.000: /BUS_UNIT:: State A
#    64000.000: /BUS_UNIT:: Pre-fetch empty...
#    64000.000: /BUS_UNIT:: Fill the prefetch
#    64500.000: /BUS_UNIT:: State C
#    65000.000: /BUS_UNIT:: State C
#    65500.000: /BUS_UNIT:: state G
#    66000.000: /BUS_UNIT:: state P
#    66500.000: /BUS_UNIT:: state H
#    67000.000: /BUS_UNIT:: State A
#    67000.000: /BUS_UNIT:: Q[0] = 0x00

#    67000.000: /BUS_UNIT:: Fill the prefetch
#    67500.000: /BUS_UNIT:: State C
#    68000.000: /BUS_UNIT:: State C
#    68500.000: /BUS_UNIT:: state G
#    69000.000: /BUS_UNIT:: state P
#    69500.000: /BUS_UNIT:: state H
#    70000.000: /BUS_UNIT:: State A
#    70000.000: /BUS_UNIT:: Q[0] = 0x00  Q[1] = 0x00

#    70000.000: /BUS_UNIT:: Fill the prefetch
#    70500.000: /BUS_UNIT:: State C
#    71000.000: /BUS_UNIT:: State C
#    71500.000: /BUS_UNIT:: state G
#    72000.000: /BUS_UNIT:: state P
#    72500.000: /BUS_UNIT:: state H
#    73000.000: /BUS_UNIT:: State A
#    73000.000: /BUS_UNIT:: Q[0] = 0x00  Q[1] = 0x00  Q[2] = 0x00

#    73500.000: /BUS_UNIT:: State A
#    73500.000: /BUS_UNIT:: Q[0] = 0x00  Q[1] = 0x00  Q[2] = 0x00

#    74000.000: /BUS_UNIT:: State A
#    74000.000: /BUS_UNIT:: Q[0] = 0x00  Q[1] = 0x00  Q[2] = 0x00

#    74500.000: /BUS_UNIT:: State A
#    74500.000: /BUS_UNIT:: Q[0] = 0x00  Q[1] = 0x00  Q[2] = 0x00
```

```
#    75000.000: /BUS_UNIT:: State A
#    75000.000: /BUS_UNIT:: Q[0] = 0x00  Q[1] = 0x00  Q[2] = 0x00
```

```
#
# This program tests the conditional jump
# instructions. All of the jumps should be
# successful since the flags register is set
# to all true.
#
00/01;
01/08;  #ld flags #$0f
02/40;
03/0f;
04/b9;  #jmpz $07
05/07;
06/a0;  #halt
07/ba;  #jmpo  $0a
08/0a;
09/a0;  #halt
0a/bb;  #jmpc  $0d
0b/0d;
0c/a0;  #halt
0d/bc;  #jmpn  $10
0e/10;
0f/a0;  #halt
10/00;  #nop
11/a0;  #halt   (goal)
12/00;
13/00;
14/00;
15/00;


#   500.000: /BUS_UNIT:: Reset Activated
#   500.000: /BUS_UNIT:: State C
#   1000.000: /BUS_UNIT:: State C
#   1500.000: /BUS_UNIT:: state G
#   2000.000: /BUS_UNIT:: state P
#   2500.000: /BUS_UNIT:: state I
#   3000.000: /BUS_UNIT:: state O
#   3500.000: /BUS_UNIT:: state L
#   4000.000: /BUS_UNIT:: state D
#   4500.000: /BUS_UNIT:: state N
#   5000.000: /BUS_UNIT:: state N
#   5500.000: /INSTR_REG:: instr = 0x08
#   5500.000: /INSTR_REG:: instr(2:0)=0x01
#   5500.000: /INSTR_REG:: decode type = 0x00
#   5500.000: /BUS_UNIT:: state N
#   5750.000: /INSTR_REG:: instr = 0x08
#   5750.000: /INSTR_REG:: instr(2:0)=0x01
```

```
#    5750.000: /INSTR_REG:: decode type = 0x00
#    6000.000: /BUS_UNIT:: State A
#    6000.000: /BUS_UNIT:: Pre-fetch empty...
#    6000.000: /BUS_UNIT:: Fill the prefetch
#    6500.000: /BUS_UNIT:: State C
#    7000.000: /BUS_UNIT:: State C
#    7500.000: /BUS_UNIT:: state G
#    8000.000: /BUS_UNIT:: state P
#    8500.000: /BUS_UNIT:: state H
#    9000.000: /BUS_UNIT:: state J
#    9250.000: /MDR:: reading Data=0x40
#    9500.000: /MDR:: reading Data=0x40
#    9500.000: /BUS_UNIT:: state J
#    9750.000: /MDR:: reading Data=0x40
#    10000.000: /BUS_UNIT:: state J
#    10500.000: /BUS_UNIT:: state F
#    11000.000: /BUS_UNIT:: State A
#    11000.000: /BUS_UNIT:: Pre-fetch empty...
#    11000.000: /BUS_UNIT:: Fill the prefetch
#    11500.000: /BUS_UNIT:: State C
#    12000.000: /BUS_UNIT:: State C
#    12500.000: /BUS_UNIT:: state G
#    13000.000: /BUS_UNIT:: state P
#    13500.000: /BUS_UNIT:: state H
#    14000.000: /BUS_UNIT:: State A
#    14000.000: /BUS_UNIT:: Q[2] = 0x0f

#    14500.000: /BUS_UNIT:: state J
#    15000.000: /BUS_UNIT:: state J
#    15500.000: /FLAGS_REG:: Just took 1i as 0f
#    15500.000: /BUS_UNIT:: state J
#    15750.000: /FLAGS_REG:: Just took 1i as 0f
#    16000.000: /BUS_UNIT:: state J
#    16500.000: /BUS_UNIT:: state F
#    17000.000: /BUS_UNIT:: State A
#    17000.000: /BUS_UNIT:: Pre-fetch empty...
#    17000.000: /BUS_UNIT:: Fill the prefetch
#    17500.000: /BUS_UNIT:: State C
#    18000.000: /BUS_UNIT:: State C
#    18500.000: /BUS_UNIT:: state G
#    19000.000: /BUS_UNIT:: state P
#    19500.000: /BUS_UNIT:: state H
#    20000.000: /BUS_UNIT:: state J
#    20500.000: /BUS_UNIT:: state J
#    21000.000: /INSTR_REG:: instr = 0xb9
#    21000.000: /INSTR_REG:: instr(2:0)=0x07
#    21000.000: /INSTR_REG:: decode type = 0x02
```

```
#    21000.000: /BUS_UNIT:: state J
#    21250.000: /INSTR_REG:: instr = 0xb9
#    21250.000: /INSTR_REG:: instr(2:0)=0x07
#    21250.000: /INSTR_REG:: decode type = 0x02
#    21500.000: /BUS_UNIT:: state F
#    21755.000: /MDR:: reading Data=0xb9
#    21755.000: /MDR:: reading Data=0xb9
#    22000.000: /MDR:: reading Data=0xb9
#    22000.000: /BUS_UNIT:: State A
#    22000.000: /BUS_UNIT:: Pre-fetch empty...
#    22500.000: /BUS_UNIT:: State C
#    23000.000: /BUS_UNIT:: State C
#    23500.000: /BUS_UNIT:: state G
#    24000.000: /BUS_UNIT:: state P
#    24500.000: /BUS_UNIT:: state H
#    25000.000: /BUS_UNIT:: state J
#    25250.000: /MDR:: reading Data=0x07
#    25500.000: /MDR:: reading Data=0x07
#    25500.000: /BUS_UNIT:: state J
#    25750.000: /MDR:: reading Data=0x07
#    26000.000: /BUS_UNIT:: state J
#    26500.000: /BUS_UNIT:: state F
#    27000.000: /BUS_UNIT:: State A
#    27000.000: /BUS_UNIT:: Pre-fetch empty...
#    27000.000: /BUS_UNIT:: Fill the prefetch
#    27255.000: /MDR:: writing Data=0x07
#    27500.000: /MDR:: writing Data=0x07
#    27500.000: /BUS_UNIT:: State C
#    27750.000: /MDR:: writing Data=0x07
#    28000.000: /MDR:: writing Data=0x07
#    28000.000: /BUS_UNIT:: State C
#    28250.000: /MDR:: writing Data=0x07
#    28500.000: /MDR:: writing Data=0x07
#    28500.000: /BUS_UNIT:: state G
#    28750.000: /MDR:: writing Data=0x07
#    29000.000: /MDR:: writing Data=0x07
#    29000.000: /BUS_UNIT:: state P
#    29250.000: /MDR:: writing Data=0x07
#    29500.000: /MDR:: writing Data=0x07
#    29500.000: /BUS_UNIT:: state H
#    29750.000: /MDR:: writing Data=0x07
#    30000.000: /MDR:: writing Data=0x07
#    30000.000: /BUS_UNIT:: State A
#    30000.000: /BUS_UNIT:: Q[1] = 0xa0

#    30250.000: /MDR:: writing Data=0x07
#    30500.000: /MDR:: writing Data=0x07
```

```
#   30500.000: /BUS_UNIT:: state B
#   30500.000: /BUS_UNIT:: New addr = 7

#   30750.000: /MDR:: writing Data=0x07
#   31000.000: /MDR:: writing Data=0x07
#   31000.000: /BUS_UNIT:: state B
#   31000.000: /BUS_UNIT:: New addr = 7

#   31250.000: /MDR:: writing Data=0x07
#   31500.000: /BUS_UNIT:: state B
#   32000.000: /BUS_UNIT:: State A
#   32000.000: /BUS_UNIT:: Pre-fetch empty...
#   32000.000: /BUS_UNIT:: Fill the prefetch
#   32500.000: /BUS_UNIT:: State C
#   33000.000: /BUS_UNIT:: State C
#   33500.000: /BUS_UNIT:: state G
#   34000.000: /BUS_UNIT:: state P
#   34500.000: /BUS_UNIT:: state H
#   35000.000: /BUS_UNIT:: State A
#   35000.000: /BUS_UNIT:: Q[0] = 0xba

#   35500.000: /BUS_UNIT:: state J
#   36000.000: /BUS_UNIT:: state J
#   36500.000: /INSTR_REG:: instr = 0xba
#   36500.000: /INSTR_REG:: instr(2:0)=0x07
#   36500.000: /INSTR_REG:: decode type = 0x02
#   36500.000: /BUS_UNIT:: state J
#   36750.000: /INSTR_REG:: instr = 0xba
#   36750.000: /INSTR_REG:: instr(2:0)=0x07
#   36750.000: /INSTR_REG:: decode type = 0x02
#   37000.000: /BUS_UNIT:: state F
#   37255.000: /MDR:: reading Data=0xba
#   37255.000: /MDR:: reading Data=0xba
#   37500.000: /MDR:: reading Data=0xba
#   37500.000: /BUS_UNIT:: State A
#   37500.000: /BUS_UNIT:: Pre-fetch empty...
#   38000.000: /BUS_UNIT:: State C
#   38500.000: /BUS_UNIT:: State C
#   39000.000: /BUS_UNIT:: state G
#   39500.000: /BUS_UNIT:: state P
#   40000.000: /BUS_UNIT:: state H
#   40500.000: /BUS_UNIT:: state J
#   40750.000: /MDR:: reading Data=0x0a
#   41000.000: /MDR:: reading Data=0x0a
#   41000.000: /BUS_UNIT:: state J
#   41250.000: /MDR:: reading Data=0x0a
#   41500.000: /BUS_UNIT:: state J
```

```
#    42000.000: /BUS_UNIT:: state F
#    42500.000: /BUS_UNIT:: State A
#    42500.000: /BUS_UNIT:: Pre-fetch empty...
#    42500.000: /BUS_UNIT:: Fill the prefetch
#    42755.000: /BUS_UNIT:: Got the default case in func switch
#    42755.000: /MDR:: writing Data=0x0a
#    43000.000: /MDR:: writing Data=0x0a
#    43000.000: /BUS_UNIT:: State C
#    43250.000: /MDR:: writing Data=0x0a
#    43500.000: /MDR:: writing Data=0x0a
#    43500.000: /BUS_UNIT:: State C
#    43750.000: /MDR:: writing Data=0x0a
#    44000.000: /MDR:: writing Data=0x0a
#    44000.000: /BUS_UNIT:: state G
#    44250.000: /MDR:: writing Data=0x0a
#    44500.000: /MDR:: writing Data=0x0a
#    44500.000: /BUS_UNIT:: state P
#    44750.000: /MDR:: writing Data=0x0a
#    45000.000: /MDR:: writing Data=0x0a
#    45000.000: /BUS_UNIT:: state H
#    45250.000: /MDR:: writing Data=0x0a
#    45500.000: /MDR:: writing Data=0x0a
#    45500.000: /BUS_UNIT:: State A
#    45500.000: /BUS_UNIT:: Q[2] = 0xa0

#    45750.000: /MDR:: writing Data=0x0a
#    46000.000: /MDR:: writing Data=0x0a
#    46000.000: /BUS_UNIT:: state B
#    46000.000: /BUS_UNIT:: New addr = a

#    46250.000: /MDR:: writing Data=0x0a
#    46500.000: /MDR:: writing Data=0x0a
#    46500.000: /BUS_UNIT:: state B
#    46500.000: /BUS_UNIT:: New addr = a

#    46750.000: /MDR:: writing Data=0x0a
#    47000.000: /BUS_UNIT:: state B
#    47500.000: /BUS_UNIT:: State A
#    47500.000: /BUS_UNIT:: Pre-fetch empty...
#    47500.000: /BUS_UNIT:: Fill the prefetch
#    48000.000: /BUS_UNIT:: State C
#    48500.000: /BUS_UNIT:: State C
#    49000.000: /BUS_UNIT:: state G
#    49500.000: /BUS_UNIT:: state P
#    50000.000: /BUS_UNIT:: state H
#    50500.000: /BUS_UNIT:: State A
#    50500.000: /BUS_UNIT:: Q[0] = 0xbb
```

```
#   51000.000: /BUS_UNIT:: state J
#   51500.000: /BUS_UNIT:: state J
#   52000.000: /INSTR_REG:: instr = 0xbb
#   52000.000: /INSTR_REG:: instr(2:0)=0x07
#   52000.000: /INSTR_REG:: decode type = 0x02
#   52000.000: /BUS_UNIT:: state J
#   52250.000: /INSTR_REG:: instr = 0xbb
#   52250.000: /INSTR_REG:: instr(2:0)=0x07
#   52250.000: /INSTR_REG:: decode type = 0x02
#   52500.000: /BUS_UNIT:: state F
#   52755.000: /MDR:: reading Data=0xbb
#   52755.000: /MDR:: reading Data=0xbb
#   53000.000: /MDR:: reading Data=0xbb
#   53000.000: /BUS_UNIT:: State A
#   53000.000: /BUS_UNIT:: Pre-fetch empty...
#   53500.000: /BUS_UNIT:: State C
#   54000.000: /BUS_UNIT:: State C
#   54500.000: /BUS_UNIT:: state G
#   55000.000: /BUS_UNIT:: state P
#   55500.000: /BUS_UNIT:: state H
#   56000.000: /BUS_UNIT:: state J
#   56250.000: /MDR:: reading Data=0x0d
#   56500.000: /MDR:: reading Data=0x0d
#   56500.000: /BUS_UNIT:: state J
#   56750.000: /MDR:: reading Data=0x0d
#   57000.000: /BUS_UNIT:: state J
#   57500.000: /BUS_UNIT:: state F
#   58000.000: /BUS_UNIT:: State A
#   58000.000: /BUS_UNIT:: Pre-fetch empty...
#   58000.000: /BUS_UNIT:: Fill the prefetch
#   58255.000: /MDR:: writing Data=0x0d
#   58500.000: /MDR:: writing Data=0x0d
#   58500.000: /BUS_UNIT:: State C
#   58750.000: /MDR:: writing Data=0x0d
#   59000.000: /MDR:: writing Data=0x0d
#   59000.000: /BUS_UNIT:: State C
#   59250.000: /MDR:: writing Data=0x0d
#   59500.000: /MDR:: writing Data=0x0d
#   59500.000: /BUS_UNIT:: state G
#   59750.000: /MDR:: writing Data=0x0d
#   60000.000: /MDR:: writing Data=0x0d
#   60000.000: /BUS_UNIT:: state P
#   60250.000: /MDR:: writing Data=0x0d
#   60500.000: /MDR:: writing Data=0x0d
#   60500.000: /BUS_UNIT:: state H
#   60750.000: /MDR:: writing Data=0x0d
#   61000.000: /MDR:: writing Data=0x0d
```

```
#    61000.000: /BUS_UNIT:: State A
#    61000.000: /BUS_UNIT:: Q[2] = 0xa0

#    61250.000: /MDR:: writing Data=0x0d
#    61500.000: /MDR:: writing Data=0x0d
#    61500.000: /BUS_UNIT:: state B
#    61500.000: /BUS_UNIT:: New addr = d

#    61750.000: /MDR:: writing Data=0x0d
#    62000.000: /MDR:: writing Data=0x0d
#    62000.000: /BUS_UNIT:: state B
#    62000.000: /BUS_UNIT:: New addr = d

#    62250.000: /MDR:: writing Data=0x0d
#    62500.000: /BUS_UNIT:: state B
#    63000.000: /BUS_UNIT:: State A
#    63000.000: /BUS_UNIT:: Pre-fetch empty...
#    63000.000: /BUS_UNIT:: Fill the prefetch
#    63500.000: /BUS_UNIT:: State C
#    64000.000: /BUS_UNIT:: State C
#    64500.000: /BUS_UNIT:: state G
#    65000.000: /BUS_UNIT:: state P
#    65500.000: /BUS_UNIT:: state H
#    66000.000: /BUS_UNIT:: State A
#    66000.000: /BUS_UNIT:: Q[0] = 0xbc

#    66500.000: /BUS_UNIT:: state J
#    67000.000: /BUS_UNIT:: state J
#    67500.000: /INSTR_REG:: instr = 0xbc
#    67500.000: /INSTR_REG:: instr(2:0)=0x07
#    67500.000: /INSTR_REG:: decode type = 0x02
#    67500.000: /BUS_UNIT:: state J
#    67750.000: /INSTR_REG:: instr = 0xbc
#    67750.000: /INSTR_REG:: instr(2:0)=0x07
#    67750.000: /INSTR_REG:: decode type = 0x02
#    68000.000: /BUS_UNIT:: state F
#    68255.000: /MDR:: reading Data=0xbc
#    68255.000: /MDR:: reading Data=0xbc
#    68500.000: /MDR:: reading Data=0xbc
#    68500.000: /BUS_UNIT:: State A
#    68500.000: /BUS_UNIT:: Pre-fetch empty...
#    69000.000: /BUS_UNIT:: State C
#    69500.000: /BUS_UNIT:: State C
#    70000.000: /BUS_UNIT:: state G
#    70500.000: /BUS_UNIT:: state P
#    71000.000: /BUS_UNIT:: state H
#    71500.000: /BUS_UNIT:: state J
```

```
#   71750.000: /MDR:: reading Data=0x10
#   72000.000: /MDR:: reading Data=0x10
#   72000.000: /BUS_UNIT:: state J
#   72250.000: /MDR:: reading Data=0x10
#   72500.000: /BUS_UNIT:: state J
#   73000.000: /BUS_UNIT:: state F
#   73500.000: /BUS_UNIT:: State A
#   73500.000: /BUS_UNIT:: Pre-fetch empty...
#   73500.000: /BUS_UNIT:: Fill the prefetch
#   73755.000: /MDR:: writing Data=0x10
#   74000.000: /MDR:: writing Data=0x10
#   74000.000: /BUS_UNIT:: State C
#   74250.000: /MDR:: writing Data=0x10
#   74500.000: /MDR:: writing Data=0x10
#   74500.000: /BUS_UNIT:: State C
#   74750.000: /MDR:: writing Data=0x10
#   75000.000: /MDR:: writing Data=0x10
#   75000.000: /BUS_UNIT:: state G
#   75250.000: /MDR:: writing Data=0x10
#   75500.000: /MDR:: writing Data=0x10
#   75500.000: /BUS_UNIT:: state P
#   75750.000: /MDR:: writing Data=0x10
#   76000.000: /MDR:: writing Data=0x10
#   76000.000: /BUS_UNIT:: state H
#   76250.000: /MDR:: writing Data=0x10
#   76500.000: /MDR:: writing Data=0x10
#   76500.000: /BUS_UNIT:: State A
#   76500.000: /BUS_UNIT:: Q[2] = 0xa0

#   76750.000: /MDR:: writing Data=0x10
#   77000.000: /MDR:: writing Data=0x10
#   77000.000: /BUS_UNIT:: state B
#   77000.000: /BUS_UNIT:: New addr = 10

#   77250.000: /MDR:: writing Data=0x10
#   77500.000: /MDR:: writing Data=0x10
#   77500.000: /BUS_UNIT:: state B
#   77500.000: /BUS_UNIT:: New addr = 10

#   77750.000: /MDR:: writing Data=0x10
#   78000.000: /BUS_UNIT:: state B
#   78500.000: /BUS_UNIT:: State A
#   78500.000: /BUS_UNIT:: Pre-fetch empty...
#   78500.000: /BUS_UNIT:: Fill the prefetch
#   79000.000: /BUS_UNIT:: State C
#   79500.000: /BUS_UNIT:: State C
#   80000.000: /BUS_UNIT:: state G
```

```
#    80500.000: /BUS_UNIT:: state P
#    81000.000: /BUS_UNIT:: state H
#    81500.000: /BUS_UNIT:: State A
#    81500.000: /BUS_UNIT:: Q[0] = 0x00

#    82000.000: /BUS_UNIT:: state J
#    82500.000: /BUS_UNIT:: state J
#    83000.000: /INSTR_REG:: instr = 0x00
#    83000.000: /INSTR_REG:: instr(2:0)=0x00
#    83000.000: /INSTR_REG:: decode type = 0x00
#    83000.000: /BUS_UNIT:: state J
#    83250.000: /INSTR_REG:: instr = 0x00
#    83250.000: /INSTR_REG:: instr(2:0)=0x00
#    83250.000: /INSTR_REG:: decode type = 0x00
#    83500.000: /BUS_UNIT:: state F
#    84000.000: /BUS_UNIT:: State A
#    84000.000: /BUS_UNIT:: Pre-fetch empty...
#    84000.000: /BUS_UNIT:: Fill the prefetch
#    84500.000: /BUS_UNIT:: State C
#    85000.000: /BUS_UNIT:: State C
#    85500.000: /BUS_UNIT:: state G
#    86000.000: /BUS_UNIT:: state P
#    86500.000: /BUS_UNIT:: state H
#    87000.000: /BUS_UNIT:: state J
#    87500.000: /BUS_UNIT:: state J
#    88000.000: /INSTR_REG:: instr = 0xa0
#    88000.000: /INSTR_REG:: instr(2:0)=0x04
#    88000.000: /INSTR_REG:: decode type = 0x02
#    88000.000: /BUS_UNIT:: state J
#    88250.000: /INSTR_REG:: instr = 0xa0
#    88250.000: /INSTR_REG:: instr(2:0)=0x04
#    88250.000: /INSTR_REG:: decode type = 0x02
#    88500.000: /BUS_UNIT:: state F
#    89000.000: /BUS_UNIT:: State A
#    89000.000: /BUS_UNIT:: Pre-fetch empty...
#    89000.000: /BUS_UNIT:: Fill the prefetch
#    89500.000: /BUS_UNIT:: State C
#    90000.000: /BUS_UNIT:: State C
#    90500.000: /BUS_UNIT:: state G
#    91000.000: /BUS_UNIT:: state P
#    91500.000: /BUS_UNIT:: state H
#    92000.000: /BUS_UNIT:: State A
#    92000.000: /BUS_UNIT:: Q[2] = 0x00

#    92000.000: /BUS_UNIT:: Fill the prefetch
#    92500.000: /BUS_UNIT:: State C
#    93000.000: /BUS_UNIT:: State C
```

```
#    93500.000: /BUS_UNIT:: state G
#    94000.000: /BUS_UNIT:: state P
#    94500.000: /BUS_UNIT:: state H
#    95000.000: /BUS_UNIT:: State A
#    95000.000: /BUS_UNIT:: Q[2] = 0x00  Q[3] = 0x00

#    95000.000: /BUS_UNIT:: Fill the prefetch
#    95500.000: /BUS_UNIT:: State C
#    96000.000: /BUS_UNIT:: State C
#    96500.000: /BUS_UNIT:: state G
#    97000.000: /BUS_UNIT:: state P
#    97500.000: /BUS_UNIT:: state H
#    98000.000: /BUS_UNIT:: State A
#    98000.000: /BUS_UNIT:: Q[2] = 0x00  Q[3] = 0x00  Q[0] = 0x00

#    98500.000: /BUS_UNIT:: State A
#    98500.000: /BUS_UNIT:: Q[2] = 0x00  Q[3] = 0x00  Q[0] = 0x00

#    99000.000: /BUS_UNIT:: State A
#    99000.000: /BUS_UNIT:: Q[2] = 0x00  Q[3] = 0x00  Q[0] = 0x00
```

```
#
# This program tests the SUB, NOT and ADD instructions.
#
#
00/01;
01/08;  #lda #$ff
02/00;
03/ff;
04/54;  #nota         a=#$00, zero=1,PSW=0x04
05/0F;
06/40;  #adda #$80    a=#$80, neg=1,PSW=0x08
07/00;
08/80;
09/48;  #suba #$70    a=#$10, zero=neg=0,PSW=0x00
0a/00;
0b/70;
0c/08;  #ldb #$ff     b=#$ff
0d/10;
0e/ff;
0f/40;  #addb #$01    b=#$00    zero=carry=1,PSW=0x05
10/10;
11/01;
12/00;
13/00;
14/00;
15/00;


#    250.000: /CONTROL_UNIT:: State A
#    500.000: /CONTROL_UNIT:: Reset activated
#    500.000: /BUS_UNIT:: Reset Activated
#    500.000: /BUS_UNIT:: State C
#    750.000: /CONTROL_UNIT:: State T
#    1000.000: /BUS_UNIT:: State C
#    1250.000: /CONTROL_UNIT:: State T
#    1500.000: /BUS_UNIT:: state G
#    1750.000: /CONTROL_UNIT:: State T
#    2000.000: /BUS_UNIT:: state P
#    2250.000: /CONTROL_UNIT:: State T
#    2500.000: /BUS_UNIT:: state I
#    2750.000: /CONTROL_UNIT:: State T
#    3000.000: /BUS_UNIT:: state O
#    3250.000: /CONTROL_UNIT:: State T
#    3500.000: /BUS_UNIT:: state L
#    3750.000: /CONTROL_UNIT:: State T
#    4000.000: /BUS_UNIT:: state D
#    4250.000: /CONTROL_UNIT:: State T
```

```
#    4500.000: /BUS_UNIT:: state N
#    4750.000: /CONTROL_UNIT:: State T
#    5000.000: /BUS_UNIT:: state N
#    5250.000: /CONTROL_UNIT:: State R
#    5500.000: /INSTR_REG:: instr = 0x08
#    5500.000: /INSTR_REG:: instr(2:0)=0x01
#    5500.000: /INSTR_REG:: decode type = 0x00
#    5500.000: /BUS_UNIT:: state N
#    5750.000: /INSTR_REG:: instr = 0x08
#    5750.000: /INSTR_REG:: instr(2:0)=0x01
#    5750.000: /INSTR_REG:: decode type = 0x00
#    5750.000: /CONTROL_UNIT:: state U
#    6000.000: /BUS_UNIT:: State A
#    6000.000: /BUS_UNIT:: Pre-fetch empty...
#    6000.000: /BUS_UNIT:: Fill the prefetch
#    6250.000: /CONTROL_UNIT:: State W
#    6500.000: /BUS_UNIT:: State C
#    6750.000: /CONTROL_UNIT:: State W
#    7000.000: /BUS_UNIT:: State C
#    7250.000: /CONTROL_UNIT:: State W
#    7500.000: /BUS_UNIT:: state G
#    7750.000: /CONTROL_UNIT:: State W
#    8000.000: /BUS_UNIT:: state P
#    8250.000: /CONTROL_UNIT:: State W
#    8500.000: /BUS_UNIT:: state H
#    8750.000: /CONTROL_UNIT:: State W
#    9000.000: /BUS_UNIT:: state J
#    9250.000: /MDR:: reading Data=0x00
#    9250.000: /CONTROL_UNIT:: State W
#    9500.000: /MDR:: reading Data=0x00
#    9500.000: /BUS_UNIT:: state J
#    9750.000: /MDR:: reading Data=0x00
#    9750.000: /CONTROL_UNIT:: State Y
#    10000.000: /BUS_UNIT:: state J
#    10250.000: /CONTROL_UNIT:: State Y
#    10500.000: /BUS_UNIT:: state F
#    10750.000: /CONTROL_UNIT:: State Y
#    11000.000: /BUS_UNIT:: State A
#    11000.000: /BUS_UNIT:: Pre-fetch empty...
#    11000.000: /BUS_UNIT:: Fill the prefetch
#    11250.000: /CONTROL_UNIT:: State V
#    11500.000: /BUS_UNIT:: State C
#    11750.000: /CONTROL_UNIT:: State V
#    12000.000: /BUS_UNIT:: State C
#    12250.000: /CONTROL_UNIT:: State V
#    12500.000: /BUS_UNIT:: state G
#    12750.000: /CONTROL_UNIT:: State V
```

```
#   13000.000: /BUS_UNIT:: state P
#   13250.000: /CONTROL_UNIT:: State V
#   13500.000: /BUS_UNIT:: state H
#   13750.000: /CONTROL_UNIT:: State V
#   14000.000: /BUS_UNIT:: State A
#   14000.000: /BUS_UNIT:: Q[2] = 0xff

#   14250.000: /CONTROL_UNIT:: State V
#   14500.000: /BUS_UNIT:: state J
#   14750.000: /CONTROL_UNIT:: State V
#   15000.000: /BUS_UNIT:: state J
#   15250.000: /CONTROL_UNIT:: State E
#   15255.000: REGA:: reading Data = 0xff
#   15500.000: REGA:: reading Data = 0xff
#   15500.000: /BUS_UNIT:: state J
#   15750.000: REGA:: reading Data = 0xff
#   15750.000: /CONTROL_UNIT:: State A
#   16000.000: /BUS_UNIT:: state J
#   16250.000: /CONTROL_UNIT:: State A
#   16500.000: /BUS_UNIT:: state F
#   16750.000: /CONTROL_UNIT:: State A
#   17000.000: /BUS_UNIT:: State A
#   17000.000: /BUS_UNIT:: Pre-fetch empty...
#   17000.000: /BUS_UNIT:: Fill the prefetch
#   17250.000: /CONTROL_UNIT:: State Q
#   17500.000: /BUS_UNIT:: State C
#   17750.000: /CONTROL_UNIT:: State Q
#   18000.000: /BUS_UNIT:: State C
#   18250.000: /CONTROL_UNIT:: State Q
#   18500.000: /BUS_UNIT:: state G
#   18750.000: /CONTROL_UNIT:: State Q
#   19000.000: /BUS_UNIT:: state P
#   19250.000: /CONTROL_UNIT:: State Q
#   19500.000: /BUS_UNIT:: state H
#   19750.000: /CONTROL_UNIT:: State Q
#   20000.000: /BUS_UNIT:: state J
#   20250.000: /CONTROL_UNIT:: State Q
#   20500.000: /BUS_UNIT:: state J
#   20750.000: /CONTROL_UNIT:: State R
#   21000.000: /INSTR_REG:: instr = 0x54
#   21000.000: /INSTR_REG:: instr(2:0)=0x02
#   21000.000: /INSTR_REG:: decode type = 0x01
#   21000.000: /BUS_UNIT:: state J
#   21250.000: /INSTR_REG:: instr = 0x54
#   21250.000: /INSTR_REG:: instr(2:0)=0x02
#   21250.000: /INSTR_REG:: decode type = 0x01
#   21250.000: /CONTROL_UNIT:: state U
```

```
#   21500.000: /BUS_UNIT:: state F
#   21750.000: /CONTROL_UNIT:: State W
#   21755.000: /MDR:: reading Data=0x54
#   22000.000: /MDR:: reading Data=0x54
#   22000.000: /BUS_UNIT:: State A
#   22000.000: /BUS_UNIT:: Pre-fetch empty...
#   22250.000: /CONTROL_UNIT:: State W
#   22500.000: /BUS_UNIT:: State C
#   22750.000: /CONTROL_UNIT:: State W
#   23000.000: /BUS_UNIT:: State C
#   23250.000: /CONTROL_UNIT:: State W
#   23500.000: /BUS_UNIT:: state G
#   23750.000: /CONTROL_UNIT:: State W
#   24000.000: /BUS_UNIT:: state P
#   24250.000: /CONTROL_UNIT:: State W
#   24500.000: /BUS_UNIT:: state H
#   24750.000: /CONTROL_UNIT:: State W
#   25000.000: /BUS_UNIT:: state J
#   25250.000: /MDR:: reading Data=0x0f
#   25250.000: /CONTROL_UNIT:: State W
#   25500.000: /MDR:: reading Data=0x0f
#   25500.000: /BUS_UNIT:: state J
#   25750.000: /MDR:: reading Data=0x0f
#   25750.000: /CONTROL_UNIT:: State Y
#   26000.000: /BUS_UNIT:: state J
#   26250.000: /CONTROL_UNIT:: State Y
#   26500.000: /BUS_UNIT:: state F
#   26750.000: /CONTROL_UNIT:: State Y
#   27000.000: /BUS_UNIT:: State A
#   27000.000: /BUS_UNIT:: Pre-fetch empty...
#   27000.000: /BUS_UNIT:: Fill the prefetch
#   27250.000: /CONTROL_UNIT:: State B
#   27255.000: REGA:: Writing 0xff on bus1
#   27500.000: REGA:: Writing 0xff on bus1
#   27500.000: /FLAGS_REG:: Just took i0 as 04
#   27500.000: /BUS_UNIT:: State C
#   27505.000: REGA:: reading Data = 0x00
#   27750.000: REGA:: reading Data = 0x00
#   27750.000: /FLAGS_REG:: Just took i0 as 04
#   27750.000: /CONTROL_UNIT:: State A
#   28000.000: /BUS_UNIT:: State C
#   28250.000: /CONTROL_UNIT:: State Q
#   28500.000: /BUS_UNIT:: state G
#   28750.000: /CONTROL_UNIT:: State Q
#   29000.000: /BUS_UNIT:: state P
#   29250.000: /CONTROL_UNIT:: State Q
#   29500.000: /BUS_UNIT:: state H
```

```
#   29750.000: /CONTROL_UNIT:: State Q
#   30000.000: /BUS_UNIT:: State A
#   30000.000: /BUS_UNIT:: Q[1] = 0x40

#   30250.000: /CONTROL_UNIT:: State Q
#   30500.000: /BUS_UNIT:: state J
#   30750.000: /CONTROL_UNIT:: State Q
#   31000.000: /BUS_UNIT:: state J
#   31250.000: /CONTROL_UNIT:: State R
#   31500.000: /INSTR_REG:: instr = 0x40
#   31500.000: /INSTR_REG:: instr(2:0)=0x00
#   31500.000: /INSTR_REG:: decode type = 0x01
#   31500.000: /BUS_UNIT:: state J
#   31750.000: /INSTR_REG:: instr = 0x40
#   31750.000: /INSTR_REG:: instr(2:0)=0x00
#   31750.000: /INSTR_REG:: decode type = 0x01
#   31750.000: /CONTROL_UNIT:: state U
#   32000.000: /BUS_UNIT:: state F
#   32250.000: /CONTROL_UNIT:: State W
#   32255.000: /MDR:: reading Data=0x40
#   32500.000: /MDR:: reading Data=0x40
#   32500.000: /BUS_UNIT:: State A
#   32500.000: /BUS_UNIT:: Pre-fetch empty...
#   32750.000: /CONTROL_UNIT:: State W
#   33000.000: /BUS_UNIT:: State C
#   33250.000: /CONTROL_UNIT:: State W
#   33500.000: /BUS_UNIT:: State C
#   33750.000: /CONTROL_UNIT:: State W
#   34000.000: /BUS_UNIT:: state G
#   34250.000: /CONTROL_UNIT:: State W
#   34500.000: /BUS_UNIT:: state P
#   34750.000: /CONTROL_UNIT:: State W
#   35000.000: /BUS_UNIT:: state H
#   35250.000: /CONTROL_UNIT:: State W
#   35500.000: /BUS_UNIT:: state J
#   35750.000: /MDR:: reading Data=0x00
#   35750.000: /CONTROL_UNIT:: State W
#   36000.000: /MDR:: reading Data=0x00
#   36000.000: /BUS_UNIT:: state J
#   36250.000: /MDR:: reading Data=0x00
#   36250.000: /CONTROL_UNIT:: State Y
#   36500.000: /BUS_UNIT:: state J
#   36750.000: /CONTROL_UNIT:: State Y
#   37000.000: /BUS_UNIT:: state F
#   37250.000: /CONTROL_UNIT:: State Y
#   37500.000: /BUS_UNIT:: State A
#   37500.000: /BUS_UNIT:: Pre-fetch empty...
```

```
#    37500.000: /BUS_UNIT:: Fill the prefetch
#    37750.000: /CONTROL_UNIT:: State V
#    38000.000: /BUS_UNIT:: State C
#    38250.000: /CONTROL_UNIT:: State V
#    38500.000: /BUS_UNIT:: State C
#    38750.000: /CONTROL_UNIT:: State V
#    39000.000: /BUS_UNIT:: state G
#    39250.000: /CONTROL_UNIT:: State V
#    39500.000: /BUS_UNIT:: state P
#    39750.000: /CONTROL_UNIT:: State V
#    40000.000: /BUS_UNIT:: state H
#    40250.000: /CONTROL_UNIT:: State V
#    40500.000: /BUS_UNIT:: State A
#    40500.000: /BUS_UNIT:: Q[3] = 0x80

#    40750.000: /CONTROL_UNIT:: State V
#    41000.000: /BUS_UNIT:: state J
#    41250.000: /CONTROL_UNIT:: State V
#    41500.000: /BUS_UNIT:: state J
#    41750.000: /CONTROL_UNIT:: State D
#    41755.000: /MDR:: reading Data=0x80
#    42000.000: /MDR:: reading Data=0x80
#    42000.000: /BUS_UNIT:: state J
#    42250.000: /MDR:: reading Data=0x80
#    42250.000: /CONTROL_UNIT:: State B
#    42255.000: /MDR:: writing Data=0x80
#    42255.000: REGA:: Writing 0x00 on bus1
#    42500.000: /MDR:: writing Data=0x80
#    42500.000: REGA:: Writing 0x00 on bus1
#    42500.000: /FLAGS_REG:: Just took i0 as 08
#    42500.000: /BUS_UNIT:: state F
#    42505.000: REGA:: reading Data = 0x80
#    42750.000: /MDR:: writing Data=0x80
#    42750.000: REGA:: reading Data = 0x80
#    42750.000: /FLAGS_REG:: Just took i0 as 08
#    42750.000: /CONTROL_UNIT:: State A
#    43000.000: /BUS_UNIT:: State A
#    43000.000: /BUS_UNIT:: Pre-fetch empty...
#    43000.000: /BUS_UNIT:: Fill the prefetch
#    43250.000: /CONTROL_UNIT:: State Q
#    43500.000: /BUS_UNIT:: State C
#    43750.000: /CONTROL_UNIT:: State Q
#    44000.000: /BUS_UNIT:: State C
#    44250.000: /CONTROL_UNIT:: State Q
#    44500.000: /BUS_UNIT:: state G
#    44750.000: /CONTROL_UNIT:: State Q
#    45000.000: /BUS_UNIT:: state P
```

```
#   45250.000: /CONTROL_UNIT:: State Q
#   45500.000: /BUS_UNIT:: state H
#   45750.000: /CONTROL_UNIT:: State Q
#   46000.000: /BUS_UNIT:: state J
#   46250.000: /CONTROL_UNIT:: State Q
#   46500.000: /BUS_UNIT:: state J
#   46750.000: /CONTROL_UNIT:: State R
#   47000.000: /INSTR_REG:: instr = 0x48
#   47000.000: /INSTR_REG:: instr(2:0)=0x01
#   47000.000: /INSTR_REG:: decode type = 0x01
#   47000.000: /BUS_UNIT:: state J
#   47250.000: /INSTR_REG:: instr = 0x48
#   47250.000: /INSTR_REG:: instr(2:0)=0x01
#   47250.000: /INSTR_REG:: decode type = 0x01
#   47250.000: /CONTROL_UNIT:: state U
#   47500.000: /BUS_UNIT:: state F
#   47750.000: /CONTROL_UNIT:: State W
#   47755.000: /MDR:: reading Data=0x48
#   48000.000: /MDR:: reading Data=0x48
#   48000.000: /BUS_UNIT:: State A
#   48000.000: /BUS_UNIT:: Pre-fetch empty...
#   48250.000: /CONTROL_UNIT:: State W
#   48500.000: /BUS_UNIT:: State C
#   48750.000: /CONTROL_UNIT:: State W
#   49000.000: /BUS_UNIT:: State C
#   49250.000: /CONTROL_UNIT:: State W
#   49500.000: /BUS_UNIT:: state G
#   49750.000: /CONTROL_UNIT:: State W
#   50000.000: /BUS_UNIT:: state P
#   50250.000: /CONTROL_UNIT:: State W
#   50500.000: /BUS_UNIT:: state H
#   50750.000: /CONTROL_UNIT:: State W
#   51000.000: /BUS_UNIT:: state J
#   51250.000: /MDR:: reading Data=0x00
#   51250.000: /CONTROL_UNIT:: State W
#   51500.000: /MDR:: reading Data=0x00
#   51500.000: /BUS_UNIT:: state J
#   51750.000: /MDR:: reading Data=0x00
#   51750.000: /CONTROL_UNIT:: State Y
#   52000.000: /BUS_UNIT:: state J
#   52250.000: /CONTROL_UNIT:: State Y
#   52500.000: /BUS_UNIT:: state F
#   52750.000: /CONTROL_UNIT:: State Y
#   53000.000: /BUS_UNIT:: State A
#   53000.000: /BUS_UNIT:: Pre-fetch empty...
#   53000.000: /BUS_UNIT:: Fill the prefetch
#   53250.000: /CONTROL_UNIT:: State V
```

```
#    53500.000: /BUS_UNIT:: State C
#    53750.000: /CONTROL_UNIT:: State V
#    54000.000: /BUS_UNIT:: State C
#    54250.000: /CONTROL_UNIT:: State V
#    54500.000: /BUS_UNIT:: state G
#    54750.000: /CONTROL_UNIT:: State V
#    55000.000: /BUS_UNIT:: state P
#    55250.000: /CONTROL_UNIT:: State V
#    55500.000: /BUS_UNIT:: state H
#    55750.000: /CONTROL_UNIT:: State V
#    56000.000: /BUS_UNIT:: State A
#    56000.000: /BUS_UNIT:: Q[2] = 0x70

#    56250.000: /CONTROL_UNIT:: State V
#    56500.000: /BUS_UNIT:: state J
#    56750.000: /CONTROL_UNIT:: State V
#    57000.000: /BUS_UNIT:: state J
#    57250.000: /CONTROL_UNIT:: State D
#    57255.000: /MDR:: reading Data=0x70
#    57500.000: /MDR:: reading Data=0x70
#    57500.000: /BUS_UNIT:: state J
#    57750.000: /MDR:: reading Data=0x70
#    57750.000: /CONTROL_UNIT:: State B
#    57755.000: /MDR:: writing Data=0x70
#    57755.000: REGA:: Writing 0x80 on bus1
#    58000.000: /MDR:: writing Data=0x70
#    58000.000: REGA:: Writing 0x80 on bus1
#    58000.000: /FLAGS_REG:: Just took i0 as 00
#    58000.000: /BUS_UNIT:: state F
#    58005.000: REGA:: reading Data = 0x10
#    58250.000: /MDR:: writing Data=0x70
#    58250.000: REGA:: reading Data = 0x10
#    58250.000: /FLAGS_REG:: Just took i0 as 00
#    58250.000: /CONTROL_UNIT:: State A
#    58500.000: /BUS_UNIT:: State A
#    58500.000: /BUS_UNIT:: Pre-fetch empty...
#    58500.000: /BUS_UNIT:: Fill the prefetch
#    58750.000: /CONTROL_UNIT:: State Q
#    59000.000: /BUS_UNIT:: State C
#    59250.000: /CONTROL_UNIT:: State Q
#    59500.000: /BUS_UNIT:: State C
#    59750.000: /CONTROL_UNIT:: State Q
#    60000.000: /BUS_UNIT:: state G
#    60250.000: /CONTROL_UNIT:: State Q
#    60500.000: /BUS_UNIT:: state P
#    60750.000: /CONTROL_UNIT:: State Q
#    61000.000: /BUS_UNIT:: state H
```

```
#    61250.000: /CONTROL_UNIT:: State Q
#    61500.000: /BUS_UNIT:: state J
#    61750.000: /CONTROL_UNIT:: State Q
#    62000.000: /BUS_UNIT:: state J
#    62250.000: /CONTROL_UNIT:: State R
#    62500.000: /INSTR_REG:: instr = 0x08
#    62500.000: /INSTR_REG:: instr(2:0)=0x01
#    62500.000: /INSTR_REG:: decode type = 0x00
#    62500.000: /BUS_UNIT:: state J
#    62750.000: /INSTR_REG:: instr = 0x08
#    62750.000: /INSTR_REG:: instr(2:0)=0x01
#    62750.000: /INSTR_REG:: decode type = 0x00
#    62750.000: /CONTROL_UNIT:: state U
#    63000.000: /BUS_UNIT:: state F
#    63250.000: /CONTROL_UNIT:: State W
#    63255.000: /MDR:: reading Data=0x08
#    63500.000: /MDR:: reading Data=0x08
#    63500.000: /BUS_UNIT:: State A
#    63500.000: /BUS_UNIT:: Pre-fetch empty...
#    63750.000: /CONTROL_UNIT:: State W
#    64000.000: /BUS_UNIT:: State C
#    64250.000: /CONTROL_UNIT:: State W
#    64500.000: /BUS_UNIT:: State C
#    64750.000: /CONTROL_UNIT:: State W
#    65000.000: /BUS_UNIT:: state G
#    65250.000: /CONTROL_UNIT:: State W
#    65500.000: /BUS_UNIT:: state P
#    65750.000: /CONTROL_UNIT:: State W
#    66000.000: /BUS_UNIT:: state H
#    66250.000: /CONTROL_UNIT:: State W
#    66500.000: /BUS_UNIT:: state J
#    66750.000: /MDR:: reading Data=0x10
#    66750.000: /CONTROL_UNIT:: State W
#    67000.000: /MDR:: reading Data=0x10
#    67000.000: /BUS_UNIT:: state J
#    67250.000: /MDR:: reading Data=0x10
#    67250.000: /CONTROL_UNIT:: State Y
#    67500.000: /BUS_UNIT:: state J
#    67750.000: /CONTROL_UNIT:: State Y
#    68000.000: /BUS_UNIT:: state F
#    68250.000: /CONTROL_UNIT:: State Y
#    68500.000: /BUS_UNIT:: State A
#    68500.000: /BUS_UNIT:: Pre-fetch empty...
#    68500.000: /BUS_UNIT:: Fill the prefetch
#    68750.000: /CONTROL_UNIT:: State V
#    69000.000: /BUS_UNIT:: State C
#    69250.000: /CONTROL_UNIT:: State V
```

```
#    69500.000: /BUS_UNIT:: State C
#    69750.000: /CONTROL_UNIT:: State V
#    70000.000: /BUS_UNIT:: state G
#    70250.000: /CONTROL_UNIT:: State V
#    70500.000: /BUS_UNIT:: state P
#    70750.000: /CONTROL_UNIT:: State V
#    71000.000: /BUS_UNIT:: state H
#    71250.000: /CONTROL_UNIT:: State V
#    71500.000: /BUS_UNIT:: State A
#    71500.000: /BUS_UNIT:: Q[1] = 0xff

#    71750.000: /CONTROL_UNIT:: State V
#    72000.000: /BUS_UNIT:: state J
#    72250.000: /CONTROL_UNIT:: State V
#    72500.000: /BUS_UNIT:: state J
#    72750.000: /CONTROL_UNIT:: State E
#    72755.000: REGB:: Data = 0xff
#    73000.000: REGB:: Data = 0xff
#    73000.000: /BUS_UNIT:: state J
#    73250.000: REGB:: Data = 0xff
#    73250.000: /CONTROL_UNIT:: State A
#    73500.000: /BUS_UNIT:: state J
#    73750.000: /CONTROL_UNIT:: State A
#    74000.000: /BUS_UNIT:: state F
#    74250.000: /CONTROL_UNIT:: State A
#    74500.000: /BUS_UNIT:: State A
#    74500.000: /BUS_UNIT:: Pre-fetch empty...
#    74500.000: /BUS_UNIT:: Fill the prefetch
#    74750.000: /CONTROL_UNIT:: State Q
#    75000.000: /BUS_UNIT:: State C
#    75250.000: /CONTROL_UNIT:: State Q
#    75500.000: /BUS_UNIT:: State C
#    75750.000: /CONTROL_UNIT:: State Q
#    76000.000: /BUS_UNIT:: state G
#    76250.000: /CONTROL_UNIT:: State Q
#    76500.000: /BUS_UNIT:: state P
#    76750.000: /CONTROL_UNIT:: State Q
#    77000.000: /BUS_UNIT:: state H
#    77250.000: /CONTROL_UNIT:: State Q
#    77500.000: /BUS_UNIT:: state J
#    77750.000: /CONTROL_UNIT:: State Q
#    78000.000: /BUS_UNIT:: state J
#    78250.000: /CONTROL_UNIT:: State R
#    78500.000: /INSTR_REG:: instr = 0x40
#    78500.000: /INSTR_REG:: instr(2:0)=0x00
#    78500.000: /INSTR_REG:: decode type = 0x01
#    78500.000: /BUS_UNIT:: state J
```

```
#   78750.000: /INSTR_REG:: instr = 0x40
#   78750.000: /INSTR_REG:: instr(2:0)=0x00
#   78750.000: /INSTR_REG:: decode type = 0x01
#   78750.000: /CONTROL_UNIT:: state U
#   79000.000: /BUS_UNIT:: state F
#   79250.000: /CONTROL_UNIT:: State W
#   79255.000: /MDR:: reading Data=0x40
#   79500.000: /MDR:: reading Data=0x40
#   79500.000: /BUS_UNIT:: State A
#   79500.000: /BUS_UNIT:: Pre-fetch empty...
#   79750.000: /CONTROL_UNIT:: State W
#   80000.000: /BUS_UNIT:: State C
#   80250.000: /CONTROL_UNIT:: State W
#   80500.000: /BUS_UNIT:: State C
#   80750.000: /CONTROL_UNIT:: State W
#   81000.000: /BUS_UNIT:: state G
#   81250.000: /CONTROL_UNIT:: State W
#   81500.000: /BUS_UNIT:: state P
#   81750.000: /CONTROL_UNIT:: State W
#   82000.000: /BUS_UNIT:: state H
#   82250.000: /CONTROL_UNIT:: State W
#   82500.000: /BUS_UNIT:: state J
#   82750.000: /MDR:: reading Data=0x10
#   82750.000: /CONTROL_UNIT:: State W
#   83000.000: /MDR:: reading Data=0x10
#   83000.000: /BUS_UNIT:: state J
#   83250.000: /MDR:: reading Data=0x10
#   83250.000: /CONTROL_UNIT:: State Y
#   83500.000: /BUS_UNIT:: state J
#   83750.000: /CONTROL_UNIT:: State Y
#   84000.000: /BUS_UNIT:: state F
#   84250.000: /CONTROL_UNIT:: State Y
#   84500.000: /BUS_UNIT:: State A
#   84500.000: /BUS_UNIT:: Pre-fetch empty...
#   84500.000: /BUS_UNIT:: Fill the prefetch
#   84750.000: /CONTROL_UNIT:: State V
#   85000.000: /BUS_UNIT:: State C
#   85250.000: /CONTROL_UNIT:: State V
#   85500.000: /BUS_UNIT:: State C
#   85750.000: /CONTROL_UNIT:: State V
#   86000.000: /BUS_UNIT:: state G
#   86250.000: /CONTROL_UNIT:: State V
#   86500.000: /BUS_UNIT:: state P
#   86750.000: /CONTROL_UNIT:: State V
#   87000.000: /BUS_UNIT:: state H
#   87250.000: /CONTROL_UNIT:: State V
#   87500.000: /BUS_UNIT:: State A
```

```
#    87500.000: /BUS_UNIT:: Q[0] = 0x01

#    87750.000: /CONTROL_UNIT:: State V
#    88000.000: /BUS_UNIT:: state J
#    88250.000: /CONTROL_UNIT:: State V
#    88500.000: /BUS_UNIT:: state J
#    88750.000: /CONTROL_UNIT:: State D
#    88755.000: /MDR:: reading Data=0x01
#    89000.000: /MDR:: reading Data=0x01
#    89000.000: /BUS_UNIT:: state J
#    89250.000: /MDR:: reading Data=0x01
#    89250.000: /CONTROL_UNIT:: State B
#    89255.000: /MDR:: writing Data=0x01
#    89255.000: REGB:: Writing 0xff on bus1
#    89500.000: /MDR:: writing Data=0x01
#    89500.000: /FLAGS_REG:: Just took i0 as 05
#    89500.000: REGB:: Writing 0xff on bus1
#    89500.000: /BUS_UNIT:: state F
#    89505.000: REGB:: Data = 0x00
#    89750.000: /MDR:: writing Data=0x01
#    89750.000: /FLAGS_REG:: Just took i0 as 05
#    89750.000: REGB:: Data = 0x00
#    89750.000: /CONTROL_UNIT:: State A
#    90000.000: /BUS_UNIT:: State A
#    90000.000: /BUS_UNIT:: Pre-fetch empty...
#    90000.000: /BUS_UNIT:: Fill the prefetch
#    90250.000: /CONTROL_UNIT:: State Q
#    90500.000: /BUS_UNIT:: State C
#    90750.000: /CONTROL_UNIT:: State Q
#    91000.000: /BUS_UNIT:: State C
#    91250.000: /CONTROL_UNIT:: State Q
#    91500.000: /BUS_UNIT:: state G
#    91750.000: /CONTROL_UNIT:: State Q
#    92000.000: /BUS_UNIT:: state P
#    92250.000: /CONTROL_UNIT:: State Q
#    92500.000: /BUS_UNIT:: state H
#    92750.000: /CONTROL_UNIT:: State Q
#    93000.000: /BUS_UNIT:: state J
#    93250.000: /CONTROL_UNIT:: State Q
#    93500.000: /BUS_UNIT:: state J
#    93750.000: /CONTROL_UNIT:: State R
#    94000.000: /INSTR_REG:: instr = 0x00
#    94000.000: /INSTR_REG:: instr(2:0)=0x00
#    94000.000: /INSTR_REG:: decode type = 0x00
#    94000.000: /BUS_UNIT:: state J
#    94250.000: /INSTR_REG:: instr = 0x00
#    94250.000: /INSTR_REG:: instr(2:0)=0x00
      .
```

```
#    94250.000: /INSTR_REG:: decode type = 0x00
#    94250.000: /CONTROL_UNIT:: state U
#    94250.000: /CONTROL_UNIT:: Nop..next state = A
```

```
#
# This program is used to test the OR, XOR and AND instructions
#
00/01;
01/08;  #ldb  #$00
02/10;
03/00;
04/60;  #orb #$ff     b=#$ff, neg=1,PSW=0x08
05/10;
06/ff;
07/68;  #xorb #$55    b=#$aa, neg=1,PSW=0x08
08/10;
09/55;
0a/58;  #andb #$44    b=#$00, zero=1,PSW=0x04
0b/10;
0c/44;
0d/08;  #ldc #$00     c=#$00
0e/20;
0f/00;
10/68;  #xorc #$aa    c=$#aa ,neg=1,PSW=0x08
11/20;
12/aa;
13/00;
14/00;
15/00;
16/00;
17/00;


#    250.000: /CONTROL_UNIT:: State A
#    500.000: /CONTROL_UNIT:: Reset activated
#    500.000: /BUS_UNIT:: Reset Activated
#    500.000: /BUS_UNIT:: State C
#    750.000: /CONTROL_UNIT:: State T
#    1000.000: /BUS_UNIT:: State C
#    1250.000: /CONTROL_UNIT:: State T
#    1500.000: /BUS_UNIT:: state G
#    1750.000: /CONTROL_UNIT:: State T
#    2000.000: /BUS_UNIT:: state P
#    2250.000: /CONTROL_UNIT:: State T
#    2500.000: /BUS_UNIT:: state I
#    2750.000: /CONTROL_UNIT:: State T
#    3000.000: /BUS_UNIT:: state O
#    3250.000: /CONTROL_UNIT:: State T
#    3500.000: /BUS_UNIT:: state L
#    3750.000: /CONTROL_UNIT:: State T
#    4000.000: /BUS_UNIT:: state D
```

```
#    4250.000: /CONTROL_UNIT:: State T
#    4500.000: /BUS_UNIT:: state N
#    4750.000: /CONTROL_UNIT:: State T
#    5000.000: /BUS_UNIT:: state N
#    5250.000: /CONTROL_UNIT:: State R
#    5500.000: /INSTR_REG:: instr = 0x08
#    5500.000: /INSTR_REG:: instr(2:0)=0x01
#    5500.000: /INSTR_REG:: decode type = 0x00
#    5500.000: /BUS_UNIT:: state N
#    5750.000: /INSTR_REG:: instr = 0x08
#    5750.000: /INSTR_REG:: instr(2:0)=0x01
#    5750.000: /INSTR_REG:: decode type = 0x00
#    5750.000: /CONTROL_UNIT:: state U
#    6000.000: /BUS_UNIT:: State A
#    6000.000: /BUS_UNIT:: Pre-fetch empty...
#    6000.000: /BUS_UNIT:: Fill the prefetch
#    6250.000: /CONTROL_UNIT:: State W
#    6500.000: /BUS_UNIT:: State C
#    6750.000: /CONTROL_UNIT:: State W
#    7000.000: /BUS_UNIT:: State C
#    7250.000: /CONTROL_UNIT:: State W
#    7500.000: /BUS_UNIT:: state G
#    7750.000: /CONTROL_UNIT:: State W
#    8000.000: /BUS_UNIT:: state P
#    8250.000: /CONTROL_UNIT:: State W
#    8500.000: /BUS_UNIT:: state H
#    8750.000: /CONTROL_UNIT:: State W
#    9000.000: /BUS_UNIT:: state J
#    9250.000: /MDR:: reading Data=0x10
#    9250.000: /CONTROL_UNIT:: State W
#    9500.000: /MDR:: reading Data=0x10
#    9500.000: /BUS_UNIT:: state J
#    9750.000: /MDR:: reading Data=0x10
#    9750.000: /CONTROL_UNIT:: State Y
#    10000.000: /BUS_UNIT:: state J
#    10250.000: /CONTROL_UNIT:: State Y
#    10500.000: /BUS_UNIT:: state F
#    10750.000: /CONTROL_UNIT:: State Y
#    11000.000: /BUS_UNIT:: State A
#    11000.000: /BUS_UNIT:: Pre-fetch empty...
#    11000.000: /BUS_UNIT:: Fill the prefetch
#    11250.000: /CONTROL_UNIT:: State V
#    11500.000: /BUS_UNIT:: State C
#    11750.000: /CONTROL_UNIT:: State V
#    12000.000: /BUS_UNIT:: State C
#    12250.000: /CONTROL_UNIT:: State V
#    12500.000: /BUS_UNIT:: state G
```

```
#    12750.000: /CONTROL_UNIT:: State V
#    13000.000: /BUS_UNIT:: state P
#    13250.000: /CONTROL_UNIT:: State V
#    13500.000: /BUS_UNIT:: state H
#    13750.000: /CONTROL_UNIT:: State V
#    14000.000: /BUS_UNIT:: State A
#    14000.000: /BUS_UNIT:: Q[2] = 0x00

#    14250.000: /CONTROL_UNIT:: State V
#    14500.000: /BUS_UNIT:: state J
#    14750.000: /CONTROL_UNIT:: State V
#    15000.000: /BUS_UNIT:: state J
#    15250.000: /CONTROL_UNIT:: State E
#    15255.000: REGB:: Data = 0x00
#    15500.000: REGB:: Data = 0x00
#    15500.000: /BUS_UNIT:: state J
#    15750.000: REGB:: Data = 0x00
#    15750.000: /CONTROL_UNIT:: State A
#    16000.000: /BUS_UNIT:: state J
#    16250.000: /CONTROL_UNIT:: State A
#    16500.000: /BUS_UNIT:: state F
#    16750.000: /CONTROL_UNIT:: State A
#    17000.000: /BUS_UNIT:: State A
#    17000.000: /BUS_UNIT:: Pre-fetch empty...
#    17000.000: /BUS_UNIT:: Fill the prefetch
#    17250.000: /CONTROL_UNIT:: State Q
#    17500.000: /BUS_UNIT:: State C
#    17750.000: /CONTROL_UNIT:: State Q
#    18000.000: /BUS_UNIT:: State C
#    18250.000: /CONTROL_UNIT:: State Q
#    18500.000: /BUS_UNIT:: state G
#    18750.000: /CONTROL_UNIT:: State Q
#    19000.000: /BUS_UNIT:: state P
#    19250.000: /CONTROL_UNIT:: State Q
#    19500.000: /BUS_UNIT:: state H
#    19750.000: /CONTROL_UNIT:: State Q
#    20000.000: /BUS_UNIT:: state J
#    20250.000: /CONTROL_UNIT:: State Q
#    20500.000: /BUS_UNIT:: state J
#    20750.000: /CONTROL_UNIT:: State R
#    21000.000: /INSTR_REG:: instr = 0x60
#    21000.000: /INSTR_REG:: instr(2:0)=0x04
#    21000.000: /INSTR_REG:: decode type = 0x01
#    21000.000: /BUS_UNIT:: state J
#    21250.000: /INSTR_REG:: instr = 0x60
#    21250.000: /INSTR_REG:: instr(2:0)=0x04
#    21250.000: /INSTR_REG:: decode type = 0x01
```

```
#   21250.000: /CONTROL_UNIT:: state U
#   21500.000: /BUS_UNIT:: state F
#   21750.000: /CONTROL_UNIT:: State W
#   21755.000: /MDR:: reading Data=0x60
#   22000.000: /MDR:: reading Data=0x60
#   22000.000: /BUS_UNIT:: State A
#   22000.000: /BUS_UNIT:: Pre-fetch empty...
#   22250.000: /CONTROL_UNIT:: State W
#   22500.000: /BUS_UNIT:: State C
#   22750.000: /CONTROL_UNIT:: State W
#   23000.000: /BUS_UNIT:: State C
#   23250.000: /CONTROL_UNIT:: State W
#   23500.000: /BUS_UNIT:: state G
#   23750.000: /CONTROL_UNIT:: State W
#   24000.000: /BUS_UNIT:: state P
#   24250.000: /CONTROL_UNIT:: State W
#   24500.000: /BUS_UNIT:: state H
#   24750.000: /CONTROL_UNIT:: State W
#   25000.000: /BUS_UNIT:: state J
#   25250.000: /MDR:: reading Data=0x10
#   25250.000: /CONTROL_UNIT:: State W
#   25500.000: /MDR:: reading Data=0x10
#   25500.000: /BUS_UNIT:: state J
#   25750.000: /MDR:: reading Data=0x10
#   25750.000: /CONTROL_UNIT:: State Y
#   26000.000: /BUS_UNIT:: state J
#   26250.000: /CONTROL_UNIT:: State Y
#   26500.000: /BUS_UNIT:: state F
#   26750.000: /CONTROL_UNIT:: State Y
#   27000.000: /BUS_UNIT:: State A
#   27000.000: /BUS_UNIT:: Pre-fetch empty...
#   27000.000: /BUS_UNIT:: Fill the prefetch
#   27250.000: /CONTROL_UNIT:: State V
#   27500.000: /BUS_UNIT:: State C
#   27750.000: /CONTROL_UNIT:: State V
#   28000.000: /BUS_UNIT:: State C
#   28250.000: /CONTROL_UNIT:: State V
#   28500.000: /BUS_UNIT:: state G
#   28750.000: /CONTROL_UNIT:: State V
#   29000.000: /BUS_UNIT:: state P
#   29250.000: /CONTROL_UNIT:: State V
#   29500.000: /BUS_UNIT:: state H
#   29750.000: /CONTROL_UNIT:: State V
#   30000.000: /BUS_UNIT:: State A
#   30000.000: /BUS_UNIT:: Q[1] = 0xff

#   30250.000: /CONTROL_UNIT:: State V
```

```
#    30500.000: /BUS_UNIT:: state J
#    30750.000: /CONTROL_UNIT:: State V
#    31000.000: /BUS_UNIT:: state J
#    31250.000: /CONTROL_UNIT:: State D
#    31255.000: /MDR:: reading Data=0xff
#    31500.000: /MDR:: reading Data=0xff
#    31500.000: /BUS_UNIT:: state J
#    31750.000: /MDR:: reading Data=0xff
#    31750.000: /CONTROL_UNIT:: State B
#    31755.000: /MDR:: writing Data=0xff
#    31755.000: REGB:: Writing 0x00 on bus1
#    32000.000: /MDR:: writing Data=0xff
#    32000.000: /FLAGS_REG:: Just took i0 as 08
#    32000.000: REGB:: Writing 0x00 on bus1
#    32000.000: /BUS_UNIT:: state F
#    32005.000: REGB:: Data = 0xff
#    32250.000: /MDR:: writing Data=0xff
#    32250.000: /FLAGS_REG:: Just took i0 as 08
#    32250.000: REGB:: Data = 0xff
#    32250.000: /CONTROL_UNIT:: State A
#    32500.000: /BUS_UNIT:: State A
#    32500.000: /BUS_UNIT:: Pre-fetch empty...
#    32500.000: /BUS_UNIT:: Fill the prefetch
#    32750.000: /CONTROL_UNIT:: State Q
#    33000.000: /BUS_UNIT:: State C
#    33250.000: /CONTROL_UNIT:: State Q
#    33500.000: /BUS_UNIT:: State C
#    33750.000: /CONTROL_UNIT:: State Q
#    34000.000: /BUS_UNIT:: state G
#    34250.000: /CONTROL_UNIT:: State Q
#    34500.000: /BUS_UNIT:: state P
#    34750.000: /CONTROL_UNIT:: State Q
#    35000.000: /BUS_UNIT:: state H
#    35250.000: /CONTROL_UNIT:: State Q
#    35500.000: /BUS_UNIT:: state J
#    35750.000: /CONTROL_UNIT:: State Q
#    36000.000: /BUS_UNIT:: state J
#    36250.000: /CONTROL_UNIT:: State R
#    36500.000: /INSTR_REG:: instr = 0x68
#    36500.000: /INSTR_REG:: instr(2:0)=0x05
#    36500.000: /INSTR_REG:: decode type = 0x01
#    36500.000: /BUS_UNIT:: state J
#    36750.000: /INSTR_REG:: instr = 0x68
#    36750.000: /INSTR_REG:: instr(2:0)=0x05
#    36750.000: /INSTR_REG:: decode type = 0x01
#    36750.000: /CONTROL_UNIT:: state U
#    37000.000: /BUS_UNIT:: state F
```

```
#    37250.000: /CONTROL_UNIT:: State W
#    37255.000: /MDR:: reading Data=0x68
#    37500.000: /MDR:: reading Data=0x68
#    37500.000: /BUS_UNIT:: State A
#    37500.000: /BUS_UNIT:: Pre-fetch empty...
#    37750.000: /CONTROL_UNIT:: State W
#    38000.000: /BUS_UNIT:: State C
#    38250.000: /CONTROL_UNIT:: State W
#    38500.000: /BUS_UNIT:: State C
#    38750.000: /CONTROL_UNIT:: State W
#    39000.000: /BUS_UNIT:: state G
#    39250.000: /CONTROL_UNIT:: State W
#    39500.000: /BUS_UNIT:: state P
#    39750.000: /CONTROL_UNIT:: State W
#    40000.000: /BUS_UNIT:: state H
#    40250.000: /CONTROL_UNIT:: State W
#    40500.000: /BUS_UNIT:: state J
#    40750.000: /MDR:: reading Data=0x10
#    40750.000: /CONTROL_UNIT:: State W
#    41000.000: /MDR:: reading Data=0x10
#    41000.000: /BUS_UNIT:: state J
#    41250.000: /MDR:: reading Data=0x10
#    41250.000: /CONTROL_UNIT:: State Y
#    41500.000: /BUS_UNIT:: state J
#    41750.000: /CONTROL_UNIT:: State Y
#    42000.000: /BUS_UNIT:: state F
#    42250.000: /CONTROL_UNIT:: State Y
#    42500.000: /BUS_UNIT:: State A
#    42500.000: /BUS_UNIT:: Pre-fetch empty...
#    42500.000: /BUS_UNIT:: Fill the prefetch
#    42750.000: /CONTROL_UNIT:: State V
#    43000.000: /BUS_UNIT:: State C
#    43250.000: /CONTROL_UNIT:: State V
#    43500.000: /BUS_UNIT:: State C
#    43750.000: /CONTROL_UNIT:: State V
#    44000.000: /BUS_UNIT:: state G
#    44250.000: /CONTROL_UNIT:: State V
#    44500.000: /BUS_UNIT:: state P
#    44750.000: /CONTROL_UNIT:: State V
#    45000.000: /BUS_UNIT:: state H
#    45250.000: /CONTROL_UNIT:: State V
#    45500.000: /BUS_UNIT:: State A
#    45500.000: /BUS_UNIT:: Q[0] = 0x55

#    45750.000: /CONTROL_UNIT:: State V
#    46000.000: /BUS_UNIT:: state J
#    46250.000: /CONTROL_UNIT:: State V
```

```
#    46500.000: /BUS_UNIT:: state J
#    46750.000: /CONTROL_UNIT:: State D
#    46755.000: /MDR:: reading Data=0x55
#    47000.000: /MDR:: reading Data=0x55
#    47000.000: /BUS_UNIT:: state J
#    47250.000: /MDR:: reading Data=0x55
#    47250.000: /CONTROL_UNIT:: State B
#    47255.000: /MDR:: writing Data=0x55
#    47255.000: REGB:: Writing 0xff on bus1
#    47500.000: /MDR:: writing Data=0x55
#    47500.000: /FLAGS_REG:: Just took i0 as 08
#    47500.000: REGB:: Writing 0xff on bus1
#    47500.000: /BUS_UNIT:: state F
#    47505.000: REGB:: Data = 0xaa
#    47750.000: /MDR:: writing Data=0x55
#    47750.000: /FLAGS_REG:: Just took i0 as 08
#    47750.000: REGB:: Data = 0xaa
#    47750.000: /CONTROL_UNIT:: State A
#    48000.000: /BUS_UNIT:: State A
#    48000.000: /BUS_UNIT:: Pre-fetch empty...
#    48000.000: /BUS_UNIT:: Fill the prefetch
#    48250.000: /CONTROL_UNIT:: State Q
#    48500.000: /BUS_UNIT:: State C
#    48750.000: /CONTROL_UNIT:: State Q
#    49000.000: /BUS_UNIT:: State C
#    49250.000: /CONTROL_UNIT:: State Q
#    49500.000: /BUS_UNIT:: state G
#    49750.000: /CONTROL_UNIT:: State Q
#    50000.000: /BUS_UNIT:: state P
#    50250.000: /CONTROL_UNIT:: State Q
#    50500.000: /BUS_UNIT:: state H
#    50750.000: /CONTROL_UNIT:: State Q
#    51000.000: /BUS_UNIT:: state J
#    51250.000: /CONTROL_UNIT:: State Q
#    51500.000: /BUS_UNIT:: state J
#    51750.000: /CONTROL_UNIT:: State R
#    52000.000: /INSTR_REG:: instr = 0x58
#    52000.000: /INSTR_REG:: instr(2:0)=0x03
#    52000.000: /INSTR_REG:: decode type = 0x01
#    52000.000: /BUS_UNIT:: state J
#    52250.000: /INSTR_REG:: instr = 0x58
#    52250.000: /INSTR_REG:: instr(2:0)=0x03
#    52250.000: /INSTR_REG:: decode type = 0x01
#    52250.000: /CONTROL_UNIT:: state U
#    52500.000: /BUS_UNIT:: state F
#    52750.000: /CONTROL_UNIT:: State W
#    52755.000: /MDR:: reading Data=0x58
```

```
#   53000.000: /MDR:: reading Data=0x58
#   53000.000: /BUS_UNIT:: State A
#   53000.000: /BUS_UNIT:: Pre-fetch empty...
#   53250.000: /CONTROL_UNIT:: State W
#   53500.000: /BUS_UNIT:: State C
#   53750.000: /CONTROL_UNIT:: State W
#   54000.000: /BUS_UNIT:: State C
#   54250.000: /CONTROL_UNIT:: State W
#   54500.000: /BUS_UNIT:: state G
#   54750.000: /CONTROL_UNIT:: State W
#   55000.000: /BUS_UNIT:: state P
#   55250.000: /CONTROL_UNIT:: State W
#   55500.000: /BUS_UNIT:: state H
#   55750.000: /CONTROL_UNIT:: State W
#   56000.000: /BUS_UNIT:: state J
#   56250.000: /MDR:: reading Data=0x10
#   56250.000: /CONTROL_UNIT:: State W
#   56500.000: /MDR:: reading Data=0x10
#   56500.000: /BUS_UNIT:: state J
#   56750.000: /MDR:: reading Data=0x10
#   56750.000: /CONTROL_UNIT:: State Y
#   57000.000: /BUS_UNIT:: state J
#   57250.000: /CONTROL_UNIT:: State Y
#   57500.000: /BUS_UNIT:: state F
#   57750.000: /CONTROL_UNIT:: State Y
#   58000.000: /BUS_UNIT:: State A
#   58000.000: /BUS_UNIT:: Pre-fetch empty...
#   58000.000: /BUS_UNIT:: Fill the prefetch
#   58250.000: /CONTROL_UNIT:: State V
#   58500.000: /BUS_UNIT:: State C
#   58750.000: /CONTROL_UNIT:: State V
#   59000.000: /BUS_UNIT:: State C
#   59250.000: /CONTROL_UNIT:: State V
#   59500.000: /BUS_UNIT:: state G
#   59750.000: /CONTROL_UNIT:: State V
#   60000.000: /BUS_UNIT:: state P
#   60250.000: /CONTROL_UNIT:: State V
#   60500.000: /BUS_UNIT:: state H
#   60750.000: /CONTROL_UNIT:: State V
#   61000.000: /BUS_UNIT:: State A
#   61000.000: /BUS_UNIT:: Q[3] = 0x44

#   61250.000: /CONTROL_UNIT:: State V
#   61500.000: /BUS_UNIT:: state J
#   61750.000: /CONTROL_UNIT:: State V
#   62000.000: /BUS_UNIT:: state J
#   62250.000: /CONTROL_UNIT:: State D
```

```
#   62255.000: /MDR:: reading Data=0x44
#   62500.000: /MDR:: reading Data=0x44
#   62500.000: /BUS_UNIT:: state J
#   62750.000: /MDR:: reading Data=0x44
#   62750.000: /CONTROL_UNIT:: State B
#   62755.000: /MDR:: writing Data=0x44
#   62755.000: REGB:: Writing 0xaa on bus1
#   63000.000: /MDR:: writing Data=0x44
#   63000.000: /FLAGS_REG:: Just took i0 as 04
#   63000.000: REGB:: Writing 0xaa on bus1
#   63000.000: /BUS_UNIT:: state F
#   63005.000: REGB:: Data = 0x00
#   63250.000: /MDR:: writing Data=0x44
#   63250.000: /FLAGS_REG:: Just took i0 as 04
#   63250.000: REGB:: Data = 0x00
#   63250.000: /CONTROL_UNIT:: State A
#   63500.000: /BUS_UNIT:: State A
#   63500.000: /BUS_UNIT:: Pre-fetch empty...
#   63500.000: /BUS_UNIT:: Fill the prefetch
#   63750.000: /CONTROL_UNIT:: State Q
#   64000.000: /BUS_UNIT:: State C
#   64250.000: /CONTROL_UNIT:: State Q
#   64500.000: /BUS_UNIT:: State C
#   64750.000: /CONTROL_UNIT:: State Q
#   65000.000: /BUS_UNIT:: state G
#   65250.000: /CONTROL_UNIT:: State Q
#   65500.000: /BUS_UNIT:: state P
#   65750.000: /CONTROL_UNIT:: State Q
#   66000.000: /BUS_UNIT:: state H
#   66250.000: /CONTROL_UNIT:: State Q
#   66500.000: /BUS_UNIT:: state J
#   66750.000: /CONTROL_UNIT:: State Q
#   67000.000: /BUS_UNIT:: state J
#   67250.000: /CONTROL_UNIT:: State R
#   67500.000: /INSTR_REG:: instr = 0x08
#   67500.000: /INSTR_REG:: instr(2:0)=0x01
#   67500.000: /INSTR_REG:: decode type = 0x00
#   67500.000: /BUS_UNIT:: state J
#   67750.000: /INSTR_REG:: instr = 0x08
#   67750.000: /INSTR_REG:: instr(2:0)=0x01
#   67750.000: /INSTR_REG:: decode type = 0x00
#   67750.000: /CONTROL_UNIT:: state U
#   68000.000: /BUS_UNIT:: state F
#   68250.000: /CONTROL_UNIT:: State W
#   68255.000: /MDR:: reading Data=0x08
#   68500.000: /MDR:: reading Data=0x08
#   68500.000: /BUS_UNIT:: State A
```

```
#    68500.000: /BUS_UNIT:: Pre-fetch empty...
#    68750.000: /CONTROL_UNIT:: State W
#    69000.000: /BUS_UNIT:: State C
#    69250.000: /CONTROL_UNIT:: State W
#    69500.000: /BUS_UNIT:: State C
#    69750.000: /CONTROL_UNIT:: State W
#    70000.000: /BUS_UNIT:: state G
#    70250.000: /CONTROL_UNIT:: State W
#    70500.000: /BUS_UNIT:: state P
#    70750.000: /CONTROL_UNIT:: State W
#    71000.000: /BUS_UNIT:: state H
#    71250.000: /CONTROL_UNIT:: State W
#    71500.000: /BUS_UNIT:: state J
#    71750.000: /MDR:: reading Data=0x20
#    71750.000: /CONTROL_UNIT:: State W
#    72000.000: /MDR:: reading Data=0x20
#    72000.000: /BUS_UNIT:: state J
#    72250.000: /MDR:: reading Data=0x20
#    72250.000: /CONTROL_UNIT:: State Y
#    72500.000: /BUS_UNIT:: state J
#    72750.000: /CONTROL_UNIT:: State Y
#    73000.000: /BUS_UNIT:: state F
#    73250.000: /CONTROL_UNIT:: State Y
#    73500.000: /BUS_UNIT:: State A
#    73500.000: /BUS_UNIT:: Pre-fetch empty...
#    73500.000: /BUS_UNIT:: Fill the prefetch
#    73750.000: /CONTROL_UNIT:: State V
#    74000.000: /BUS_UNIT:: State C
#    74250.000: /CONTROL_UNIT:: State V
#    74500.000: /BUS_UNIT:: State C
#    74750.000: /CONTROL_UNIT:: State V
#    75000.000: /BUS_UNIT:: state G
#    75250.000: /CONTROL_UNIT:: State V
#    75500.000: /BUS_UNIT:: state P
#    75750.000: /CONTROL_UNIT:: State V
#    76000.000: /BUS_UNIT:: state H
#    76250.000: /CONTROL_UNIT:: State V
#    76500.000: /BUS_UNIT:: State A
#    76500.000: /BUS_UNIT:: Q[2] = 0x00

#    76750.000: /CONTROL_UNIT:: State V
#    77000.000: /BUS_UNIT:: state J
#    77250.000: /CONTROL_UNIT:: State V
#    77500.000: /BUS_UNIT:: state J
#    77750.000: /CONTROL_UNIT:: State E
#    77755.000: REGC:: reading Data = 0x00
#    78000.000: REGC:: reading Data = 0x00
```

```
#   78000.000: /BUS_UNIT:: state J
#   78250.000: REGC:: reading Data = 0x00
#   78250.000: /CONTROL_UNIT:: State A
#   78500.000: /BUS_UNIT:: state J
#   78750.000: /CONTROL_UNIT:: State A
#   79000.000: /BUS_UNIT:: state F
#   79250.000: /CONTROL_UNIT:: State A
#   79500.000: /BUS_UNIT:: State A
#   79500.000: /BUS_UNIT:: Pre-fetch empty...
#   79500.000: /BUS_UNIT:: Fill the prefetch
#   79750.000: /CONTROL_UNIT:: State Q
#   80000.000: /BUS_UNIT:: State C
#   80250.000: /CONTROL_UNIT:: State Q
#   80500.000: /BUS_UNIT:: State C
#   80750.000: /CONTROL_UNIT:: State Q
#   81000.000: /BUS_UNIT:: state G
#   81250.000: /CONTROL_UNIT:: State Q
#   81500.000: /BUS_UNIT:: state P
#   81750.000: /CONTROL_UNIT:: State Q
#   82000.000: /BUS_UNIT:: state H
#   82250.000: /CONTROL_UNIT:: State Q
#   82500.000: /BUS_UNIT:: state J
#   82750.000: /CONTROL_UNIT:: State Q
#   83000.000: /BUS_UNIT:: state J
#   83250.000: /CONTROL_UNIT:: State R
#   83500.000: /INSTR_REG:: instr = 0x68
#   83500.000: /INSTR_REG:: instr(2:0)=0x05
#   83500.000: /INSTR_REG:: decode type = 0x01
#   83500.000: /BUS_UNIT:: state J
#   83750.000: /INSTR_REG:: instr = 0x68
#   83750.000: /INSTR_REG:: instr(2:0)=0x05
#   83750.000: /INSTR_REG:: decode type = 0x01
#   83750.000: /CONTROL_UNIT:: state U
#   84000.000: /BUS_UNIT:: state F
#   84250.000: /CONTROL_UNIT:: State W
#   84255.000: /MDR:: reading Data=0x68
#   84500.000: /MDR:: reading Data=0x68
#   84500.000: /BUS_UNIT:: State A
#   84500.000: /BUS_UNIT:: Pre-fetch empty...
#   84750.000: /CONTROL_UNIT:: State W
#   85000.000: /BUS_UNIT:: State C
#   85250.000: /CONTROL_UNIT:: State W
#   85500.000: /BUS_UNIT:: State C
#   85750.000: /CONTROL_UNIT:: State W
#   86000.000: /BUS_UNIT:: state G
#   86250.000: /CONTROL_UNIT:: State W
#   86500.000: /BUS_UNIT:: state P
```

```
#   86750.000: /CONTROL_UNIT:: State W
#   87000.000: /BUS_UNIT:: state H
#   87250.000: /CONTROL_UNIT:: State W
#   87500.000: /BUS_UNIT:: state J
#   87750.000: /MDR:: reading Data=0x20
#   87750.000: /CONTROL_UNIT:: State W
#   88000.000: /MDR:: reading Data=0x20
#   88000.000: /BUS_UNIT:: state J
#   88250.000: /MDR:: reading Data=0x20
#   88250.000: /CONTROL_UNIT:: State Y
#   88500.000: /BUS_UNIT:: state J
#   88750.000: /CONTROL_UNIT:: State Y
#   89000.000: /BUS_UNIT:: state F
#   89250.000: /CONTROL_UNIT:: State Y
#   89500.000: /BUS_UNIT:: State A
#   89500.000: /BUS_UNIT:: Pre-fetch empty...
#   89500.000: /BUS_UNIT:: Fill the prefetch
#   89750.000: /CONTROL_UNIT:: State V
#   90000.000: /BUS_UNIT:: State C
#   90250.000: /CONTROL_UNIT:: State V
#   90500.000: /BUS_UNIT:: State C
#   90750.000: /CONTROL_UNIT:: State V
#   91000.000: /BUS_UNIT:: state G
#   91250.000: /CONTROL_UNIT:: State V
#   91500.000: /BUS_UNIT:: state P
#   91750.000: /CONTROL_UNIT:: State V
#   92000.000: /BUS_UNIT:: state H
#   92250.000: /CONTROL_UNIT:: State V
#   92500.000: /BUS_UNIT:: State A
#   92500.000: /BUS_UNIT:: Q[1] = 0xaa

#   92750.000: /CONTROL_UNIT:: State V
#   93000.000: /BUS_UNIT:: state J
#   93250.000: /CONTROL_UNIT:: State V
#   93500.000: /BUS_UNIT:: state J
#   93750.000: /CONTROL_UNIT:: State D
#   93755.000: /MDR:: reading Data=0xaa
#   94000.000: /MDR:: reading Data=0xaa
#   94000.000: /BUS_UNIT:: state J
#   94250.000: /MDR:: reading Data=0xaa
#   94250.000: /CONTROL_UNIT:: State B
#   94255.000: /MDR:: writing Data=0xaa
#   94255.000: REGC:: Writing 0x00 on bus1
#   94500.000: /MDR:: writing Data=0xaa
#   94500.000: /FLAGS_REG:: Just took i0 as 08
#   94500.000: REGC:: Writing 0x00 on bus1
#   94500.000: /BUS_UNIT:: state F
```

```
#   94505.000: REGC:: reading Data = 0xaa
#   94750.000: /MDR:: writing Data=0xaa
#   94750.000: /FLAGS_REG:: Just took i0 as 08
#   94750.000: REGC:: reading Data = 0xaa
#   94750.000: /CONTROL_UNIT:: State A
#   95000.000: /BUS_UNIT:: State A
#   95000.000: /BUS_UNIT:: Pre-fetch empty...
#   95000.000: /BUS_UNIT:: Fill the prefetch
#   95250.000: /CONTROL_UNIT:: State Q
#   95500.000: /BUS_UNIT:: State C
#   95750.000: /CONTROL_UNIT:: State Q
#   96000.000: /BUS_UNIT:: State C
#   96250.000: /CONTROL_UNIT:: State Q
#   96500.000: /BUS_UNIT:: state G
#   96750.000: /CONTROL_UNIT:: State Q
#   97000.000: /BUS_UNIT:: state P
#   97250.000: /CONTROL_UNIT:: State Q
#   97500.000: /BUS_UNIT:: state H
#   97750.000: /CONTROL_UNIT:: State Q
#   98000.000: /BUS_UNIT:: state J
#   98250.000: /CONTROL_UNIT:: State Q
#   98500.000: /BUS_UNIT:: state J
#   98750.000: /CONTROL_UNIT:: State R
#   99000.000: /INSTR_REG:: instr = 0x00
#   99000.000: /INSTR_REG:: instr(2:0)=0x00
#   99000.000: /INSTR_REG:: decode type = 0x00
#   99000.000: /BUS_UNIT:: state J
#   99250.000: /INSTR_REG:: instr = 0x00
#   99250.000: /INSTR_REG:: instr(2:0)=0x00
#   99250.000: /INSTR_REG:: decode type = 0x00
#   99250.000: /CONTROL_UNIT:: state U
#   99250.000: /CONTROL_UNIT:: Nop..next state = A
```